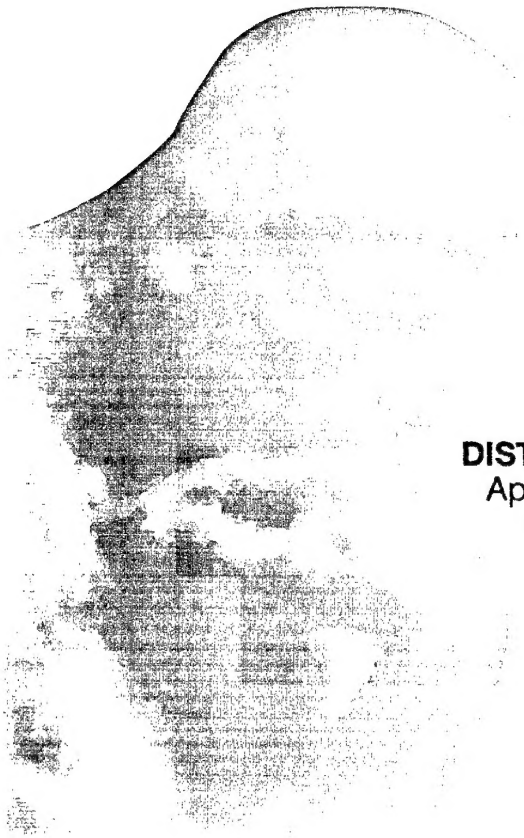
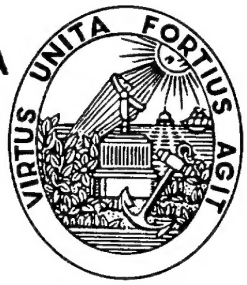
VECPAR

and parallel processing

Proceedings

Part I (June 21)





Faculdade de Engenharia
da Universidade do Porto

2000 June, 21 22 23

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	12 January 2001	Conference Proceedings

4. TITLE AND SUBTITLE

VECPAR - 4th International Meeting on Vector and Parallel Processing

Part 1

5. FUNDING NUMBERS

F61775-00-WF071

6. AUTHOR(S)

Conference Committee

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

FEUP-FACULDADE DE ENGENHARIA DA Universidade do Porto
RUA DOS BRAGAS
Porto 4050-123
Portugal

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

EOARD
PSC 802 BOX 14
FPO 09499-0200

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

CSP 00-5071

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

A

13. ABSTRACT (Maximum 200 words)

The Final Proceedings for VECPAR - 4th International Meeting on Vector and Parallel Processing, 21 June 2000 - 23 June 2000, an interdisciplinary conference covering topics in all areas of vector, parallel and distributed computing applied to a broad range of research disciplines with a focus on engineering. The principal topics include: Cellular Automata, Computational Fluid Dynamics, Crash and Structural Analysis, Data Warehousing and Data Mining, Distributed Computing and Operating Systems, Fault Tolerant Systems, Imaging and Graphics, Interconnection Networks, Languages and Tools, Numerical Methods, Parallel and Distributed Algorithms, Real-time and Embedded Systems, Reconfigurable Systems, Linear Algebra Algorithms and Software for Large Scientific Problems, Computer Organization, Image Analysis and Synthesis, and Nonlinear Problems.

14. SUBJECT TERMS

EOARD, Modelling & Simulation, Parallel Computing, Distributed Computing

15. NUMBER OF PAGES

Three volumes:
1016 pages total
(plus TOC, and front matter)

16. PRICE CODE

N/A

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

VECPAR'2000

4rd International Meeting on Vector and Parallel Processing

2000, June 21-23

Conference Proceedings
Part I
(Wednesday, June 21)

FEUP
Faculdade de Engenharia
da Universidade do Porto

Preface

This book is part of a 3-volume set with the written versions of all invited talks, papers and posters presented at *VECPAR'2000 - 4th International Meeting on Vector and Parallel Processing.*

The Preface and the Table of Contents are identical in all 3 volumes (one for each day of the conference), numbered in sequence. Papers are grouped according to the session where they were presented.

The conference programme added up to a total of 6 plenary and 20 parallel sessions, comprising 6 invited talks, 66 papers and 11 posters.

It is our great pleasure to express our gratitude to all people that helped us during the preparation of this event. The expertise provided by the Scientific Committee was crucial in the selection of more than 100 abstracts submitted for possible presentation.

Even at the risk of forgetting some people, we would like to express our gratitude to the following people, whose collaboration went well beyond the call of duty. Fernando Jorge and Vítor Carvalho, for creation and maintenance of the conference web page; Alice Silva for the secretarial work; Dr. Jaime Villate for his assistance in organisational matters; and Nuno Sousa and Alberto Mota, for authoring the procedure for abstract submission via web.

Porto, June 2000 *The Organising and Scientific Committee Chairs*

VECPAR'2000 was held at Fundação Dr. António Cupertino de Miranda, in Porto (Portugal), from 21 to 23 June, 2000.

VECPAR is a series of conferences, on vector and parallel computing organised by the Faculty of Engineering of the University of Porto (FEUP) since 1993.

Committees

Organising Committee

A. Augusto de Sousa (Chair)
José Couto Marques (Co-chair)
José Magalhães Cruz

Local Advisory Committee

Carlos Costa
Raimundo Delgado
José Marques dos Santos
Fernando Nunes Ferreira
Lígia Ribeiro
José Silva Matos
Paulo Tavares de Castro
Raul Vidal

Scientific Committee

J. Palma (Chair)	Univ. do Porto, Portugal
J. Dongarra (Co-chair)	Univ. of Tennessee and Oak Ridge National Lab., USA
V. Hernandez (Co-chair)	Univ. Politécnica de Valencia, Spain
P. Amestoy	ENSEEIHT-IRIT, Toulouse, France
T. Barth	NASA Ames Research Center, USA
A. Campilho	Univ. do Porto, Portugal
G. Candler	Univ. of Minnesota, USA
A. Chalmers	Univ. of Bristol, England
B. Chapman	Univ. of Southampton, England
A. Coutinho	Univ. Federal do Rio de Janeiro, Brazil
J. C. Cunha	Univ. Nova de Lisboa, Portugal
F. d'Almeida	Univ. do Porto, Portugal
M. Daydé	ENSEEIHT-IRIT, Toulouse, France
J. Dekeyser	Univ. des Sciences et Technologies, Lille, France
P. Devloo	Univ. Estadual de Campinas (UNICAMP), Brazil
J. Duarte	Univ. do Porto, Portugal
I. Duff	Rutherford Appleton Lab., England, and CERFACS, France
D. Falcão	Univ. Federal do Rio de Janeiro, Brazil
J. Fortes	Purdue Univ., USA
S. Gama	Univ. do Porto, Portugal
M. Giles	Univ. of Oxford, England
L. Giraud	CERFACS, France
G. Golub	Stanford Univ., USA
D. Heermann	Univ. Heidelberg,Germany
W. Janke	Univ. of Leipzig, Germany
M. Kamel	Univ. of Waterloo, Canada
M.-T. Kechadi	Univ. College Dublin, Ireland
D. Knight	Rutgers-State Univ. of New Jersey, USA
V. Kumar	Univ. of Minnesota, USA
R. Lohner	George Mason Univ., USA
E. Luque	Univ. Autónoma de Barcelona, Spain
J. Macedo	Univ. do Porto, Portugal
P. Marquet	Univ. des Sciences et Technologies, Lille, France
P. de Miguel	Univ. Politécnica de Madrid, Spain
F. Moura	Univ. do Minho, Portugal
E. Oñate	Univ. Politécnica de Catalunya, Spain
A. Padilha	Univ. do Porto, Portugal
R. Pandey	Univ. of Southern Mississipi, USA
M. Peric	Technische Univ. Hamburg-Harrurg, Germany
T. Priol	IRISA/INRIA, France
R. Ralha	Univ. do Minho, Portugal
M. Ruano	Univ. do Algarve, Portugal
D. Ruiz	ENSEEIHT-IRIT, Toulouse, France
H. Ruskin	Dublin City Univ., Ireland
J. G. Silva	Univ. de Coimbra, Portugal
F. Tirado	Univ. Complutense, Spain
B. Tourancheau	Univ. Claude Bernard de Lyon, France
M. Valero	Univ. Politécnica de Catalunya, Spain
A. van der Steen	Utrecht Univ., The Netherlands
J. Vuillemin	École Normale Superieure, Paris, France
J.-S. Wang	National Univ. of Singapore, Singapore
P. Watson	Univ. of Newscastle, England
P. Welch	Univ. of Kent at Canterbury, England
E. Zapata	Univ. de Malaga, Spain

Sponsoring Organisations

The Organising Committee is very grateful to all sponsoring organisations for their support:

FEUP - Faculdade de Engenharia da Universidade do Porto
UP - Universidade do Porto

CMP - Câmara Municipal do Porto
EOARD - European Office of Aerospace Research and Development
FACM - Fundação Dr. António Cupertino de Miranda
FCCN - Fundação para a Computação Científica Nacional
FCG - Fundação Calouste Gulbenkian
FCT - Fundação para a Ciência e a Tecnologia
FLAD - Fundação Luso-Americana para o Desenvolvimento
ICCTI/BC - Inst. de Cooperação Científica e Tecnológica Internacional/British Council
INESC Porto - Instituto de Engenharia de Sistemas e de Computadores do Porto
OE - Ordem dos Engenheiros
Porto Convention Bureau

ALCATEL
CISCO Systems
COMPAQ
MICROSOFT
NEC European Supercomputer Systems
NORTEL Networks
SIEMENS

PART I (June 21, Wednesday)

Session 8: Linear Algebra

June 22. Thursday (Room A. 9:00-10:20)

Invited Talk

(June 22. Thursday. Auditorium. 10:50-11:50)

Posters

The poster session will be held simultaneously with the Industrial session.
(June 22. Thursday. Entrance Hall. 11:50-12:50)

Computational Grids*

Ian Foster
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439

Carl Kesselman
Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292

In this introductory chapter, we lay the groundwork for the rest of the book by providing a more detailed picture of the expected purpose, shape, and architecture of future grid systems. We structure the chapter in terms of six questions that we believe are central to this discussion: Why do we need computational grids? What types of applications will grids be used for? Who will use grids? How will grids be used? What is involved in building a grid? And, what problems must be solved to make grids commonplace? We provide an overview of each of these issues here, referring to subsequent chapters for more detailed discussion.

1 Reasons for Computational Grids

Why do we need computational grids? Computational approaches to problem solving have proven their worth in almost every field of human endeavor. Computers are used for modeling and simulating complex scientific and engineering problems, diagnosing medical conditions, controlling industrial equipment, forecasting the weather, managing stock portfolios, and many other purposes. Yet, although there are certainly challenging problems that exceed our ability to solve them, computers are still used much less extensively than they could be. To pick just one example, university researchers make extensive use of computers when studying the impact of changes in land use on biodiversity, but city planners selecting routes for new roads or planning new zoning ordinances do not. Yet it is local decisions such as these that, ultimately, shape our future.

There are a variety of reasons for this relative lack of use of computational problem-solving methods, including lack of appropriate education and tools. But one important factor is that the average computing environment remains inadequate for such computationally sophisticated purposes. While today's PC is faster than the Cray supercomputer of 10 years ago, it is still far from adequate for predicting the outcome of complex actions or selecting from among many choices. That, after all, is why supercomputers have continued to evolve.

*Reprinted by permission of Morgan Kaufmann Publishers from *The Grid: Blueprint for a Future Computing Infrastructure*, I. Foster and C. Kesselman (Eds), 1998.

1.1 Increasing Delivered Computation

We believe that the opportunity exists to provide users—whether city planners, engineers, or scientists—with substantially more computational power: an increase of three orders of magnitude within five years, and five orders of magnitude within a decade. These dramatic increases will be achieved by innovations in a wide range of areas:

1. *Technology improvement:* Evolutionary changes in VLSI technology and microprocessor architecture can be expected to result in a factor of 10 increase in computational capabilities in the next five years, and a factor of 100 increase in the next ten.

2. *Increase in demand-driven access to computational power:* Many applications have only episodic requirements for substantial computational resources. For example, a medical diagnosis system may be run only when a cardiogram is performed, a stockmarket simulation only when a user recomputes retirement benefits, or a seismic simulation only after a major earthquake. If mechanisms are in place to allow reliable, instantaneous, and transparent access to high-end resources, then from the perspective of these applications it is as if those resources are dedicated to them. Given the existence of multiteraFLOPS systems, an increase in apparent computational power of three or more orders of magnitude is feasible.

3. *Increased utilization of idle capacity:* Most low-end computers (PCs and workstations) are often idle: various studies report utilizations of around 30% in academic and commercial environments [47], [21]. Utilization can be increased by a factor of two, even for parallel programs [4], without impinging significantly on productivity. The benefit to individual users can be substantially greater: factors of 100 or 1,000 increase in peak computational capacity have been reported [41], [75].

4. *Greater sharing of computational results:* The daily weather forecast involves perhaps 10^{14} numerical operations. If we assume that the forecast is of benefit to 10^7 people, we have 10^{21} effective operations—comparable to the computation performed each day on all the world's PCs. Few other computational results or facilities are shared so effectively today, but they may be in the future as other scientific communities adopt a "big science" approach to computation. The key to more sharing may be the development of collaboratories: "... center[s] without walls, in which the nation's researchers can perform their research without regard to geographical location—interacting with colleagues, accessing instrumentation, sharing data and computational resources, and accessing information in digital libraries" [48].

5. *New problem-solving techniques and tools:* A variety of approaches can improve the efficiency or ease with which computation is applied to problem solving. For example, network-enabled solvers [17], [11] allow users to invoke advanced numerical solution methods without having to install sophisticated software. Teleimmersion techniques [50] facilitate the sharing of computational results by supporting collaborative steering of simulations and exploration of data sets.

Underlying each of these advances is the synergistic use of high-performance networking, computing, and advanced software to provide access to advanced computational capabilities, regardless of the location of users and resources.

1.2 Definition of Computational Grids

The current status of computation is analogous in some respects to that of electricity around 1910. At that time, electric power generation was possible, and new devices were being devised that depended

on electric power, but the need for each user to build and operate a new generator hindered use. The truly revolutionary development was not, in fact, electricity, but the electric power grid and the associated transmission and distribution technologies. Together, these developments provided reliable, low-cost access to a standardized service, with the result that power—which for most of human history has been accessible only in crude and not especially portable forms (human effort, horses, water power, steam engines, candles)—became universally accessible. By allowing both individuals and industries to take for granted the availability of cheap, reliable power, the electric power grid made possible both new devices and the new industries that manufactured them.

By analogy, we adopt the term *computational grid* for the infrastructure that will enable the increases in computation discussed above. A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities.

We talk about an *infrastructure* because a computational grid is concerned, above all, with large-scale pooling of resources, whether compute cycles, data, sensors, or people. Such pooling requires significant hardware infrastructure to achieve the necessary interconnections and software infrastructure to monitor and control the resulting ensemble. In the rest of this chapter, and throughout the book, we discuss in detail the nature of this infrastructure.

The need for *dependable* service is fundamental. Users require assurances that they will receive predictable, sustained, and often high levels of performance from the diverse components that constitute the grid; in the absence of such assurances, applications will not be written or used. The performance characteristics that are of interest will vary widely from application to application, but may include network bandwidth, latency, jitter, computer power, software services, security, and reliability.

The need for *consistency* of service is a second fundamental concern. As with electric power, we need standard services, accessible via standard interfaces, and operating within standard parameters. Without such standards, application development and pervasive use are impractical. A significant challenge when developing standards is to encapsulate heterogeneity without compromising high-performance execution.

Pervasive access allows us to count on services always being available, within whatever environment we expect to move. Pervasiveness does not imply that resources are everywhere or are universally accessible. We cannot access electric power in a new home until wire has been laid and an account established with the local utility; computational grids will have similarly circumscribed availability and controlled access. However, we will be able to count on universal access within the confines of whatever environment the grid is designed to support.

Finally, an infrastructure must offer *inexpensive* (relative to income) access if it is to be broadly accepted and used. Homeowners and industrialists both make use of remote billion-dollar power plants on a daily basis because the cost to them is reasonable. A computational grid must achieve similarly attractive economics.

It is the combination of dependability, consistency, and pervasiveness that will cause computational grids to have a transforming effect on how computation is performed and used. By increasing the set of capabilities that can be taken for granted to the extent that they are noticed only by their absence, grids allow new tools to be developed and widely deployed. Much as pervasive access to bitmapped displays changed our baseline assumptions for the design of application interfaces, computational grids can fundamentally change the way we think about computation and resources.

1.3 The Impact of Grids

The history of network computing shows that orders-of-magnitude improvements in underlying technology invariably enable revolutionary, often unanticipated, applications of that technology, which in

turn motivate further technological improvements. As a result, our view of network computing has undergone repeated transformations over the past 40 years.

There is considerable evidence that another such revolution is imminent. The capabilities of both computers and networks continue to increase dramatically. Ten years of research on metacomputing has created a solid base of experience in new applications that couple high-speed networking and computing. The time seems ripe for a transition from the heroic days of metacomputing to more integrated computational grids with dependable and pervasive computational capabilities and consistent interfaces. In such grids, today's metacomputing applications will be routine, and programmers will be able to explore a new generation of yet more interesting applications that leverage teraFLOP computers and petabyte storage systems interconnected by gigabit networks. We present two simple examples to illustrate how grid functionality may transform different aspects of our lives.

Today's home finance software packages leverage the pervasive availability of communication technologies such as modems, Internet service providers, and the Web to integrate up-to-date stock prices obtained from remote services into local portfolio value calculations. However, the actual computations performed on this data are relatively simple. In tomorrow's grid environment, we can imagine individuals making stock-purchasing decisions on the basis of detailed Monte Carlo analyses of future asset value, performed on remote teraFLOP computers. The instantaneous use of three orders of magnitude more computing power than today will go unnoticed by prospective retirees, but their lives will be different because of more accurate information.

Today, citizen groups evaluating a proposed new urban development must study uninspiring blueprints or perspective drawings at city hall. A computational grid will allow them to call on powerful graphics computers and databases to transform the architect's plans into realistic virtual reality depictions and to explore such design issues as energy consumption, lighting efficiency, or sound quality. Meeting online to walk through and discuss the impact of the new development on their community, they can arrive at better urban design and hence improved quality of life. Virtual reality-based simulation models of Los Angeles, produced by William Jepson, and the walkthrough model of Soda Hall at the University of California–Berkeley, constructed by Carlo Seguin and his colleagues, are interesting exemplars of this use of computing [9].

1.4 Electric Power Grids

We conclude this section by reviewing briefly some salient features of the computational grid's namesake. The electric power grid is remarkable in terms of its construction and function, which together make it one of the technological marvels of the 20th century. Within large geographical regions (e.g., North America), it forms essentially a single entity that provides power to billions of devices, in a relatively efficient, low-cost, and reliable fashion. The North American grid alone links more than ten thousand generators with billions of outlets via a complex web of physical connections and trading mechanisms [12]. The components from which the grid is constructed are highly heterogeneous in terms of their physical characteristics and are owned and operated by different organizations. Consumers differ significantly in terms of the amount of power they consume, the service guarantees they require, and the amount they are prepared to pay.

Analogies are dangerous things, and electricity is certainly very different from computation in many respects. Nevertheless, the following aspects of the power grid seem particularly relevant to the current discussion.

Importance of Economics

The role and structure of the power grid are driven to a large extent by economic factors. Oil- and coal-fired generators have significant economies of scale. A power company must be able to call upon

reserve capacity equal to its largest generator in case that generator fails: interconnections between regions allow for sharing of such reserve capacity, as well as enabling trading of excess power. The impact of economic factors on computational grids is not well understood [34]. Where and when are there economies of scale to be obtained in computational capabilities? Might economic factors lead us away from today's model of a "computer on every desktop"? We note an intriguing development. Recent advances in power generation technology (e.g., small gas turbines) and the deregulation of the power industry are leading some analysts to look to the Internet for lessons regarding the future evolution of the electric power grid!

Importance of Politics

The developers of large-scale grids tell us that their success depended on regulatory, political, and institutional developments as much as on technical innovation [12]. This lesson should be taken to heart by developers of future computational grids.

Complexity of Control

The principal technical challenges in power grids—once technology issues relating to efficient generation and high-voltage transmission had been overcome—relate to the management of a complex ensemble in which changes at a single location can have far-reaching consequences [12]. Hence, we find that the power grid includes a sophisticated infrastructure for monitoring, management, and control. Again, there appear to be many parallels between this control problem and the problem of providing performance guarantees in large-scale, dynamic, and heterogeneous computational grid environments.

2 Grid Applications

What types of applications will grids be used for? Building on experiences in gigabit testbeds [42], [59], the I-WAY network [19], and other experimental systems, we have identified five major application classes for computational grids, listed in Table 1 and described briefly in this section. More details about applications and their technical requirements are provided in the referenced chapters.

2.1 Distributed Supercomputing

Distributed supercomputing applications use grids to aggregate substantial computational resources in order to tackle problems that cannot be solved on a single system. Depending on the grid on which we are working (see Section 3), these aggregated resources might comprise the majority of the supercomputers in the country or simply all of the workstations within a company. Here are some contemporary examples:

- Distributed interactive simulation (DIS) is a technique used for training and planning in the military. Realistic scenarios may involve hundreds of thousands of entities, each with potentially complex behavior patterns. Yet even the largest current supercomputers can handle at most 20,000 entities. In recent work, researchers at the California Institute of Technology have shown how multiple supercomputers can be coupled to achieve record-breaking levels of performance.
- The accurate simulation of complex physical processes can require high spatial and temporal resolution in order to resolve fine-scale detail. Coupled supercomputers can be used in such situations to overcome resolution barriers and hence to obtain qualitatively new scientific results. Although high latencies can pose significant obstacles, coupled supercomputers have been

Category	Examples	Characteristics
Distributed supercomputing	DIS Stellar dynamics Ab initio chemistry	Very large problems needing lots of CPU, memory, etc.
High throughput	Chip design Parameter studies Cryptographic problems	Harness many otherwise idle resources to increase aggregate throughput
On demand	Medical instrumentation Network-enabled solvers Cloud detection	Remote resources integrated with local computation, often for bounded amount of time
Data intensive	Sky survey Physics data Data assimilation	Synthesis of new information from many or large data sources
Collaborative	Collaborative design Data exploration Education	Support communication or collaborative work between multiple participants

Table 1: Five major classes of grid applications.

used successfully in cosmology [54], high-resolution ab initio computational chemistry computations [52], and climate modeling [45].

Challenging issues from a grid architecture perspective include the need to coschedule what are often scarce and expensive resources, the scalability of protocols and algorithms to tens or hundreds of thousands of nodes, latency-tolerant algorithms, and achieving and maintaining high levels of performance across heterogeneous systems.

2.2 High-Throughput Computing

In high-throughput computing, the grid is used to schedule large numbers of loosely coupled or independent tasks, with the goal of putting unused processor cycles (often from idle workstations) to work. The result may be, as in distributed supercomputing, the focusing of available resources on a single problem, but the quasi-independent nature of the tasks involved leads to very different types of problems and problem-solving methods. Here are some examples:

- Platform Computing Corporation reports that the microprocessor manufacturer Advanced Micro Devices used high-throughput computing techniques to exploit over a thousand computers during the peak design phases of their K6 and K7 microprocessors. These computers are located on the desktops of AMD engineers at a number of AMD sites and were used for design verification only when not in use by engineers.
- The Condor system from the University of Wisconsin is used to manage pools of hundreds of workstations at universities and laboratories around the world [41]. These resources have been used for studies as diverse as molecular simulations of liquid crystals, studies of ground-penetrating radar, and the design of diesel engines.
- More loosely organized efforts have harnessed tens of thousands of computers distributed worldwide to tackle hard cryptographic problems [40].

2.3 On-Demand Computing

On-demand applications use grid capabilities to meet short-term requirements for resources that cannot be cost-effectively or conveniently located locally. These resources may be computation, software, data repositories, specialized sensors, and so on. In contrast to distributed supercomputing applications, these applications are often driven by cost-performance concerns rather than absolute performance. For example:

- The NEOS [17] and NetSolve [11] network-enhanced numerical solver systems allow users to couple remote software and resources into desktop applications, dispatching to remote servers calculations that are computationally demanding or that require specialized software.
- A computer-enhanced MRI machine and scanning tunneling microscope (STM) developed at the National Center for Supercomputing Applications use supercomputers to achieve realtime image processing [57], [58]. The result is a significant enhancement in the ability to understand what we are seeing and, in the case of the microscope, to steer the instrument.
- A system developed at the Aerospace Corporation for processing of data from meteorological satellites uses dynamically acquired supercomputer resources to deliver the results of a cloud detection algorithm to remote meteorologists in quasi real time [38].

The challenging issues in on-demand applications derive primarily from the dynamic nature of resource requirements and the potentially large populations of users and resources. These issues include resource location, scheduling, code management, configuration, fault tolerance, security, and payment mechanisms.

2.4 Data-Intensive Computing

In data-intensive applications, the focus is on synthesizing new information from data that is maintained in geographically distributed repositories, digital libraries, and databases. This synthesis process is often computationally and communication intensive as well.

- Future high-energy physics experiments will generate terabytes of data per day, or around a petabyte per year. The complex queries used to detect "interesting" events may need to access large fractions of this data [43]. The scientific collaborators who will access this data are widely distributed, and hence the data systems in which data is placed are likely to be distributed as well.
- The Digital Sky Survey will, ultimately, make many terabytes of astronomical photographic data available in numerous network-accessible databases. This facility enables new approaches to astronomical research based on distributed analysis, assuming that appropriate computational grid facilities exist.
- Modern meteorological forecasting systems make extensive use of data assimilation to incorporate remote satellite observations. The complete process involves the movement and processing of many gigabytes of data.

Challenging issues in data-intensive applications are the scheduling and configuration of complex, high-volume data flows through multiple levels of hierarchy.

2.5 Collaborative Computing

Collaborative applications are concerned primarily with enabling and enhancing human-to-human interactions. Such applications are often structured in terms of a virtual shared space. Many collaborative applications are concerned with enabling the shared use of computational resources such as data archives and simulations: in this case, they also have characteristics of the other application classes just described. For example:

- The BoilerMaker system developed at Argonne National Laboratory allows multiple users to collaborate on the design of emission control systems in industrial incinerators [20]. The different users interact with each other and with a simulation of the incinerator.
- The CAVE5D system supports remote, collaborative exploration of large geophysical data sets and the models that generate them—for example, a coupled physical/biological model of the Chesapeake Bay [74].
- The NICE system developed at the University of Illinois at Chicago allows children to participate in the creation and maintenance of realistic virtual worlds, for entertainment and education [60].

Challenging aspects of collaborative applications from a grid architecture perspective are the real-time requirements imposed by human perceptual capabilities and the rich variety of interactions that can take place.

We conclude this section with three general observations. First, we note that even in this brief survey we see a tremendous variety of already successful applications. This rich set has been developed despite the significant difficulties faced by programmers developing grid applications in the absence of a mature grid infrastructure. As grids evolve, we expect the range and sophistication of applications to increase dramatically. Second, we observe that almost all of the applications demonstrate a tremendous appetite for computational resources (CPU, memory, disk, etc.) that cannot be met in a timely fashion by expected growth in single-system performance. This emphasizes the importance of grid technologies as a means of sharing computation as well as a data access and communication medium. Third, we see that many of the applications are interactive, or depend on tight synchronization with computational components, and hence depend on the availability of a grid infrastructure able to provide robust performance guarantees.

3 Grid Communities

Who will use grids? One approach to understanding computational grids is to consider the communities that they serve. Because grids are above all a mechanism for sharing resources, we ask, What groups of people will have sufficient incentive to invest in the infrastructure required to enable sharing, and what resources will these communities want to share?

One perspective on these questions holds that the benefits of sharing will almost always outweigh the costs and, hence, that we will see grids that link large communities with few common interests, within which resource sharing will extend to individual PCs and workstations. If we compare a computational grid to an electric power grid, then in this view, the grid is quasi-universal, and every user has the potential to act as a cogenerator. Skeptics respond that the technical and political costs of sharing resources will rarely outweigh the benefits, especially when coupling must cross institutional boundaries. Hence, they argue that resources will be shared only when there is considerable incentive to do so: because the resource is expensive, or scarce, or because sharing enables human interactions that are otherwise difficult to achieve. In this view, grids will be specialized, designed to support specific user communities with specific goals.

Rather than take a particular position on how grids will evolve, we propose what we see as four plausible scenarios, each serving a different community. Future grids will probably include elements of all four.

3.1 Government

The first community that we consider comprises the relatively small number—thousands or perhaps tens of thousands—of officials, planners, and scientists concerned with problems traditionally assigned to national government, such as disaster response, national defense, and long-term research and planning. There can be significant advantage to applying the collective power of the nation's fastest computers, data archives, and intellect to the solution of these problems. Hence, we envision a grid that uses the fastest networks to couple relatively small numbers of high-end resources across the nation—perhaps tens of teraFLOP computers, petabytes of storage, hundreds of sites, thousands of smaller systems—for two principal purposes:

1. To provide a "strategic computing reserve," allowing substantial computing resources to be applied to large problems in times of crisis, such as to plan responses to a major environmental disaster, earthquake, or terrorist attack

2. To act as a "national collaboratory," supporting collaborative investigations of complex scientific and engineering problems, such as global change, space station design, and environmental cleanup

An important secondary benefit of this high-end national supercomputing grid is to support resource trading between the various operators of high-end resources, hence increasing the efficiency with which those resources are used.

This *national grid* is distinguished by its need to integrate diverse high-end (and hence complex) resources, the strategic importance of its overall mission, and the diversity of competing interests that must be balanced when allocating resources.

3.2 A Health Maintenance Organization

In our second example, the community supported by the grid comprises administrators and medical personnel located at a small number of hospitals within a metropolitan area. The resources to be shared are a small number of high-end computers, hundreds of workstations, administrative databases, medical image archives, and specialized instruments such as MRI machines, CAT scanners, and cardioangiography devices. The coupling of these resources into an integrated grid enables a wide range of new, computationally enhanced applications: desktop tools that use centralized supercomputer resources to run computer-aided diagnosis procedures on mammograms or to search centralized medical image archives for similar cases: life-critical applications such as telerobotic surgery and remote cardiac monitoring and analysis: auditing software that uses the many workstations across the hospital to run fraud detection algorithms on financial records: and research software that uses supercomputers and idle workstations for epidemiological research. Each of these applications exists today in research laboratories, but has rarely been deployed in ordinary hospitals because of the high cost of computation.

This *private grid* is distinguished by its relatively small scale, central management, and common purpose on the one hand, and on the other hand by the complexity inherent in using common infrastructure for both life-critical applications and less reliability-sensitive purposes and by the need to integrate low-cost commodity technologies. We can expect grids with similar characteristics to be useful in many institutions.

3.3 A Materials Science Collaboratory

The community in our third example is a group of scientists who operate and use a variety of instruments, such as electron microscopes, particle accelerators, and X-ray sources, for the characterization of materials. This community is fluid and highly distributed, comprising many hundreds of university researchers and students from around the world, in addition to the operators of the various instruments (tens of instruments, at perhaps ten centers). The resources that are being shared include the instruments themselves, data archives containing the collective knowledge of this community, sophisticated analysis software developed by different groups, and various supercomputers used for analysis. Applications enabled by this grid include remote operation of instruments, collaborative analysis, and supercomputer-based online analysis.

This *virtual grid* is characterized by a strong unifying focus and relatively narrow goals on the one hand, and on the other hand by dynamic membership, a lack of central control, and a frequent need to coexist with other uses of the same resources. We can imagine similar grids arising to meet the needs of a variety of multi-institutional research groups and multicompany virtual teams created to pursue long- or short-term goals.

3.4 Computational Market Economy

The fourth community that we consider comprises the participants in a broad-based market economy for computational services. This is a potentially enormous community with no connections beyond the usual market-oriented relationships. We can expect participants to include consumers, with their diverse needs and interests; providers of specialized services, such as financial modeling, graphics rendering, and interactive gaming; providers of compute resources; network providers, who contract to provide certain levels of network service; and various other entities such as banks and licensing organizations.

This *public grid* is in some respects the most intriguing of the four scenarios considered here, but is also the least concrete. One area of uncertainty concerns the extent to which the average consumer will also act as a producer of computational resources. The answer to this question seems to depend on two issues. Will applications emerge that can exploit loosely coupled computational resources? And, will owners of resources be motivated to contribute resources? To date, large-scale activity in this area has been limited to fairly esoteric computations—such as searching for prime numbers, breaking cryptographic codes [40], or detecting extraterrestrial communications [64]—with the benefit to the individuals being the fun of participating and the potential momentary fame if their computer solves the problem in question.

We conclude this section by noting that, in our view, each of these scenarios seems quite feasible; indeed, substantial prototypes have been created for each of the grids that we describe. Hence, we expect to see not just one single computational grid, but rather many grids, each serving a different community with its own requirements and objectives. Just which grids will evolve depends critically on three issues: the evolving economics of computing and networking, and the services that these physical infrastructure elements are used to provide; the institutional, regulatory, and political frameworks within which grids may develop; and, above all, the emergence of applications able to motivate users to invest in and use grid technologies.

4 Using Grids

How will grids be used? In metacomputing experiments conducted to date, users have been "heroic" programmers, willing to spend large amounts of time programming complex systems at a low level.

Class	Purpose	Makes use of	Concerns
End users	Solve problems	Applications	Transparency, performance
Application developers	Develop applications	Programming models, tools	Ease of use, performance
Tool developers	Develop tools, programming models	Grid services	Adaptivity, exposure of performance, security
Grid developers	Provide basic grid services	Local system services	Local simplicity, connectivity, security
System administrators	Manage grid resources	Management tools	Balancing local and global concerns

Table 2: Classes of grid users.

The resulting applications have provided compelling demonstrations of what might be, but in most cases are too expensive, unreliable, insecure, and fragile to be considered suitable for general use.

For grids to become truly useful, we need to take a significant step forward in grid programming, moving from the equivalent of assembly language to high-level languages, from one-off libraries to application toolkits, and from hand-crafted codes to shrink-wrapped applications. These goals are familiar to us from conventional programming, but in a grid environment we are faced with the additional difficulties associated with wide area operation—in particular, the need for grid applications to adapt to changes in resource properties in order to meet performance requirements. As in conventional computing, an important step toward the realization of these goals is the development of standards for applications, programming models, tools, and services, so that a division of labor can be achieved between the users and developers of different types of components.

We structure our discussion of grid tools and programming in terms of the classification illustrated in Table 2. At the lowest level, we have *grid developers*—the designers and implementors of what we might call the "Grid Protocol," by analogy with the Internet Protocol that provides the lowest-level services in the Internet—who provide the basic services required to construct a grid. Above this, we have *tool developers*, who use grid services to construct programming models and associated tools, layering higher-level services and abstractions on top of the more fundamental services provided by the grid architecture. *Application developers*, in turn, build on these programming models, tools, and services to construct grid-enabled applications for *end users* who, ideally, can use these applications without being concerned with the fact that they are operating in a grid environment. A fifth class of users, *system administrators*, is responsible for managing grid components. We now examine this model in more detail.

4.1 Grid Developers

A very small group of grid developers are responsible for implementing the basic services referred to above. We discuss the concerns encountered at this level in Section 5.

4.2 Tool Developers

Our second group of users are the developers of the tools, compilers, libraries, and so on that implement the programming models and services used by application developers. Today's small population of grid tool developers (e.g., the developers of Condor [41], Nimrod [1], NEOS [17], NetSolve [11], Horus [68],

grid-enabled implementations of the Message Passing Interface (MPI) [27], and CAVERN [39]) must build their tools on a very narrow foundation, comprising little more than the Internet Protocol. We envision that future grid systems will provide a richer set of basic services, hence making it possible to build more sophisticated and robust tools. We discuss the nature and implementation of those basic services in Section 5: briefly, they comprise versions of those services that have proven effective on today's end systems and clusters, such as authentication, process management, data access, and communication, plus new services that address specific concerns of the grid environment, such as resource location, information, fault detection, security, and electronic payment.

Tool developers must use these basic services to provide efficient implementations of the programming models that will be used by application developers. In constructing these translations, the tool developer must be concerned not only with translating the existing model to the grid environment, but also with revealing to the programmer those aspects of the grid environment that impact performance. For example, a grid-enabled MPI [27] can seek to adapt the MPI model for grid execution by incorporating specialized techniques for point-to-point and collective communication in highly heterogeneous environments: implementations of collective operations might use multicast protocols and adapt a combining tree structure in response to changing network loads. It should probably also extend the MPI model to provide programmers with access to resource location services, information about grid topology, and group communication protocols.

4.3 Application Developers

Our third class of users comprises those who construct grid-enabled applications and components. Today, these programmers write applications in what is, in effect, an assembly language: explicit calls to the Internet Protocol's User Datagram Protocol (UDP) or Transmission Control Protocol (TCP), explicit or no management of failure, hard-coded configuration decisions for specific computing systems, and so on. We are far removed from the portable, efficient, high-level languages that are used to develop sequential programs, and the advanced services that programmers can rely upon when using these languages, such as dynamic memory management and high-level I/O libraries.

Future grids will need to address the needs of application developers in two ways. They must provide *programming models* (supported by languages, libraries, and tools) that are appropriate for grid environments and a range of *services* (for security, fault detection, resource management, data access, communication, etc.) that programmers can call upon when developing applications.

The purpose of both programming models and services is to simplify thinking about and implementing complex algorithmic structures, by providing a set of abstractions that hide details unrelated to the application, while exposing design decisions that have a significant impact on program performance or correctness. In sequential programming, commonly used programming models provide us with abstractions such as subroutines and scoping: in parallel programming, we have threads and condition variables (in shared-memory parallelism), message passing, distributed arrays, and single-assignment variables. Associated services ensure that resources are allocated to processes in a reasonable fashion, provide convenient abstractions for tertiary storage, and so forth.

There is no consensus on what programming model is appropriate for a grid environment, although it seems clear that many models will be used. Table 3 summarizes some of the models that have been proposed: new models will emerge as our understanding of grid programming evolves.

As Table 3 makes clear, one approach to grid programming is to adapt models that have already proved successful in sequential or parallel environments. For example, a grid-enabled distributed shared-memory (DSM) system would support a shared-memory programming model in a grid environment, allowing programmers to specify parallelism in terms of threads and shared-memory operations. Similarly, a grid-enabled MPI would extend the popular message-passing model [27], and a

Model	Examples	Pros	Cons
Datagram/stream communication	UDP, TCP, Multicast	Low overhead	Low level
Shared memory, multithreading	POSIX Threads DSM	High level	Scalability
Data parallelism	HPF, HPC++	Automatic parallelization	Restricted applicability
Message passing	MPI, PVM	High performance	Low level
Object-oriented	CORBA, DCOM, Java RMI	Support for large-system design	Performance
Remote procedure call	DCE, ONC	Simplicity	Restricted applicability
High throughput	Condor, LSF, Nimrod	Ease of use	Restricted applicability
Group ordered	Isis, Totem	Robustness	Performance, scalability
Agents	Aglets, Telescript	Flexibility	Performance, robustness

Table 3: Potential grid programming models and their advantages and disadvantages.

grid-enabled file system would permit remote files to be accessed via the standard UNIX application programming interface (API) [66]. These approaches have the advantage of potentially allowing existing applications to be reused unchanged, but can introduce significant performance problems if the models in question do not adapt well to high-latency, dynamic, heterogeneous grid environments.

Another approach is to build on technologies that have proven effective in distributed computing, such as Remote Procedure Call (RPC) or related object-based techniques such as the Common Object Request Broker Architecture (CORBA). These technologies have significant software engineering advantages, because their encapsulation properties facilitate the modular construction of programs and the reuse of existing components. However, it remains to be seen whether these models can support performance-focused, complex applications such as teleimmersion or the construction of dynamic computations that span hundreds or thousands of processors.

The grid environment can also motivate new programming models and services. For example, high-throughput computing systems, as exemplified by Condor [41] and Nimrod [1], support problem-solving methods such as parameter studies in which complex problems are partitioned into many independent tasks. Group-ordered communication systems represent another model that is important in dynamic, unpredictable grid environments; they provide services for managing groups of processes and for delivering messages reliably to group members. Agent-based programming models represent another approach apparently well suited to grid environments: here, programs are constructed as independent entities that roam the network searching for data or performing other tasks on behalf of a user.

A wide range of new services can be expected to arise in grid environments to support the development of more complex grid applications. In addition to grid analogs of conventional services such as file systems, we will see new services for resource discovery, resource brokering, electronic payments, licensing, fault tolerance, specification of use conditions, configuration, adaptation, and distributed system management, to name just a few.

4.4 End Users

Most grid users, like most users of computers or networks today, will not write programs. Instead, they will use grid-enabled applications that make use of grid resources and services. These applications may be chemistry packages or environmental models that use grid resources for computing or data; problem-solving packages that help set up parameter study experiments [1]; mathematical packages augmented with calls to network-enabled solvers [17], [11]; or collaborative engineering packages that allow geographically separated users to cooperate on the design of complex systems.

End users typically place stringent requirements on their tools, in terms of reliability, predictability, confidentiality, and usability. The construction of applications that can meet these requirements in complex grid environments represents a major research and engineering challenge.

4.5 System Administrators

The final group of users that we consider are the system administrators who must manage the infrastructure on which computational grids operate. This task is complicated by the high degree of sharing that grids are designed to make possible. The user communities and resources associated with a particular grid will frequently span multiple administrative domains, and new services will arise—such as accounting and resource brokering—that require distributed management. Furthermore, individual resources may participate in several different grids, each with its own particular user community, access policies, and so on. For a grid to be effective, each participating resource must be administered so as to strike an appropriate balance between local policy requirements and the needs of the larger grid community. This problem has a significant political dimension, but new technical solutions are also required.

The Internet experience suggests that two keys to scalability when administering large distributed systems are to decentralize administration and to automate trans-site issues. For example, names and routes are administered locally, while essential trans-site services such as route discovery and name resolution are automated. Grids will require a new generation of tools for automatically monitoring and managing many tasks that are currently handled manually.

New administration issues that arise in grids include establishing, monitoring, and enforcing local policies in situations where the set of users may be large and dynamic; negotiating policy with other sites and users; accounting and payment mechanisms; and the establishment and management of markets and other resource-trading mechanisms. There are interesting parallels between these problems and management issues that arise in the electric power and banking industries [14], [31], [28].

5 Grid Architecture

What is involved in building a grid? To address this question, we adopt a system architect's perspective and examine the organization of the software infrastructure required to support the grid users, applications, and services discussed in the preceding sections.

As noted above, computational grids will be created to serve different communities with widely varying characteristics and requirements. Hence, it seems unlikely that we will see a single grid architecture. However, we do believe that we can identify basic services that most grids will provide, with different grids adopting different approaches to the realization of these services.

One major driver for the techniques used to implement grid services is scale. Computational infrastructure, like other infrastructures, is fractal, or self-similar at different scales. We have networks between countries, organizations, clusters, and computers; between components of a computer; and even within a single component. However, at different scales, we often operate in different physical,

Comp. model	I/O model	Resource manag.	Security
Endsystem:			
Multithreading, automatic parallelization.	Local I/O, disk-striping	Process creation OS signal delivery OS scheduling	OS kernel, hardware
Cluster (increased scale, reduced integration):			
Synchronous communication, distributed shared memory	Parallel I/O (e.g., MPI-IO), file systems	Parallel process creation, gang scheduling, OS-level signal propagation	Shared security databases
Intranet (heterogeneity, separate administration, lack of global knowledge):			
Client/server, loosely synchronous: pipelines, coupling manager/worker	Distributed file systems (DFS, HPSS), databases	Resource discovery, signal distribution networks, high throughput	Network security (Kerberos)
Internet (lack of centralized control, geographical distribution, intl. issues):			
Collaborative systems, remote control, data mining	Remote file access, digital libraries, data warehouses	Brokers, trading, mobile code negotiation	Trust delegation, public key, sandboxes

Table 4: Computer systems operating at different scales.

economic, and political regimes. For example, the access control solutions used for a laptop computer's system bus are probably not appropriate for a trans-Pacific cable.

In this section, we adopt scale as the major dimension for comparison. We consider four types of systems, of increasing scale and complexity, asking two questions for each: What new concerns does this increase in scale introduce? And how do these new concerns influence how we provide basic services? These system types are as follows (see also Table 4):

1. The *end system* provides the best model we have for what it means to compute, because it is here that most research and development efforts have focused in the past four decades.

2. The *cluster* introduces new issues of parallelism and distributed management, albeit of homogeneous systems.

3. The *intranet* introduces the additional issues of heterogeneity and geographical distribution.

4. The *internet* introduces issues associated with a lack of centralized control.

An important secondary driver for architectural solutions is the performance requirements of the grid. Stringent performance requirements amplify the effect of scale because they make it harder to hide heterogeneity. For example, if performance is not a big concern, it is straightforward to extend UNIX file I/O to support access to remote files, perhaps via a HyperText Transport Protocol (HTTP) gateway [66]. However, if performance is critical, remote access may require quite different mechanisms—such as parallel transfers over a striped network from a remote parallel file system to a local parallel computer—that are not easily expressed in terms of UNIX file I/O semantics. Hence, a

high-performance wide area grid may need to adopt quite different solutions to data access problems. In the following, we assume that we are dealing with high-performance systems; systems with lower performance requirements are generally simpler.

5.1 Basic Services

We start our discussion of architecture by reviewing the basic services provided on conventional computers. We do so because we believe that, in the absence of strong evidence to the contrary, services that have been developed and proven effective in several decades of conventional computing will also be desirable in computational grids. Grid environments also require additional services, but we claim that, to a significant extent, grid development will be concerned with extending familiar capabilities to the more complex wide area environment.

Our purpose in this subsection is not to provide a detailed exposition of well-known ideas but rather to establish a vocabulary for subsequent discussion. We assume that we are discussing a generic modern computing system, and hence refrain from prefixing each statement with "in general," "typically," and the like. Individual systems will, of course, differ from the generic systems described here, sometimes in interesting and important ways.

The first step in a computation that involves shared resources is an *authentication* process, designed to establish the identity of the user. A subsequent *authorization* process establishes the right of the user to create entities called *processes*. A process comprises one or more *threads* of control, created for either concurrency or parallelism, and executing within a *shared address space*. A process can also *communicate* with other processes via a variety of abstractions, including shared memory (with semaphores or locks), pipes, and protocols such as TCP/IP.

A user (or process acting on behalf of a user) can *control* the activities in another process—for example, to suspend, resume, or terminate its execution. This control is achieved by means of asynchronously delivered *signals*.

A process acts on behalf of its creator to *acquire resources*, by executing instructions, occupying memory, reading and writing disks, sending and receiving messages, and so on. The ability of a process to acquire resources is limited by underlying authorization mechanisms, which implement a system's *resource allocation policy*, taking into account the user's identity, prior resource consumption, and/or other criteria. *Scheduling* mechanisms in the underlying system deal with competing demands for resources and may also (for example, in realtime systems) support user requests for performance guarantees.

Underlying *accounting* mechanisms keep track of resource allocations and consumption, and *payment* mechanisms may be provided to translate resource consumption into some common currency. The underlying system will also provide *protection* mechanisms to ensure that one user's computation does not interfere with another's.

Other services provide abstractions for secondary storage. Of these, *virtual memory* is implicit, extending the shared address space abstraction already noted; *file systems* and *databases* are more explicit representations of secondary storage.

5.2 End Systems

Individual end systems—computers, storage systems, sensors, and other devices—are characterized by relatively small scale and a high degree of homogeneity and integration. There are typically just a few tens of components (processors, disks, etc.), these components are mostly of the same type, and the components and the software that controls them have been co-designed to simplify management and use and to maximize performance. (Specialized devices such as scientific instruments may be more

significantly complex, with potentially thousands of internal components, of which hundreds may be visible externally.)

Such end systems represent the simplest, and most intensively studied, environment in which to provide the services listed above. The principal challenges facing developers of future systems of this type relate to changing computer architectures (in particular, parallel architectures) and the need to integrate end systems more fully into clusters, intranets, and internets.

State of the Art

The software architectures used in conventional end systems are well known [61]. Basic services are provided by a privileged operating system, which has absolute control over the resources of the computer. This operating system handles authentication and mediates user process requests to acquire resources, communicate with other processes, access files, and so on. The integrated nature of the hardware and operating system allows high-performance implementations of important functions such as virtual memory and I/O.

Programmers develop applications for these end systems by using a variety of high-level languages and tools. A high degree of integration between processor architecture, memory system, and compiler means that high performance can often be achieved with relatively little programmer effort.

Future Directions

A significant deficiency of most end-system architectures is that they lack features necessary for integration into larger clusters, intranets, and internets. Much current research and development is concerned with evolving system end architectures in directions relevant to future computational grids. To list just three: Operating systems are evolving to support operation in clustered environments, in which services are distributed over multiple networked computers, rather than replicated on every processor [3], [65]. A second important trend is toward a greater integration of end systems (computers, disks, etc.) with networks, with the goal of reducing the overheads incurred at network interfaces and hence increasing communication rates [22], [35]. Finally, support for mobile code is starting to appear, in the form of authentication schemes, secure execution environments for downloaded code ("sandboxes"), and so on [32], [72], [71], [44].

The net effect of these various developments seems likely to be to reduce the currently sharp boundaries between end system, cluster, and intranet/internet, with the result that individual end systems will more fully embrace remote computation, as producers and/or consumers.

5.3 Clusters

The second class of systems that we consider is the cluster, or network of workstations: a collection of computers connected by a high-speed local area network and designed to be used as an integrated computing or data processing resource. A cluster, like an individual end system, is a homogeneous entity—its constituent systems differ primarily in configuration, not basic architecture—and is controlled by a single administrative entity who has complete control over each end system. The two principal complicating factors that the cluster introduces are as follows:

1. *Increased physical scale*: A cluster may comprise several hundred or thousand processors, with the result that alternative algorithms are needed for certain resource management and control functions.

2. *Reduced integration*: A desire to construct clusters from commodity parts means that clusters are often less integrated than end systems. One implication of this is reduced performance for certain functions (e.g., communication).

State of the Art

The increased scale and reduced integration of the cluster environment make the implementation of certain services more difficult and also introduce a need for new services not required in a single end system. The result tends to be either significantly reduced performance (and hence range of applications) or software architectures that modify and/or extend end-system operating systems in significant ways.

We use the problem of high-performance parallel execution to illustrate the types of issues that can arise when we seek to provide familiar end-system services in a cluster environment. In a single (multiprocessor) end system, high-performance parallel execution is typically achieved either by using specialized communication libraries such as MPI or by creating multiple threads that communicate by reading and writing a shared address space.

Both message-passing and shared-memory programming models can be implemented in a cluster. Message passing is straightforward to implement, since the commodity systems from which clusters are constructed typically support at least TCP/IP as a communication protocol. Shared memory requires additional effort: in an end system, hardware mechanisms ensure a uniform address space for all threads, but in a cluster, we are dealing with multiple address spaces. One approach to this problem is to implement a logical shared memory by providing software mechanisms for translating between local and global addresses, ensuring coherency between different versions of data, and so forth. A variety of such distributed shared-memory systems exist, varying according to the level at which sharing is permitted [76], [24], [53].

In low-performance environments, the cluster developer's job is done at this point; message-passing and DSM systems can be run as user-level programs that use conventional communication protocols and mechanisms (e.g., TCP/IP) for interprocessor communication. However, if performance is important, considerable additional development effort may be required. Conventional network protocols are orders of magnitude slower than intra-end-system communication operations. Low-latency, high-bandwidth inter-end-system communication can require modifications to the protocols used for communication, the operating system's treatment of network interfaces, or even the network interface hardware [70], [56].

The cluster developer who is concerned with parallel performance must also address the problem of coscheduling. There is little point in communicating extremely rapidly to a remote process that must be scheduled before it can respond. Coscheduling refers to techniques that seek to schedule simultaneously the processes constituting a computation on different processors [23], [63]. In certain highly integrated parallel computers, coscheduling is achieved by using a batch scheduler: processors are space shared, so that only one computation uses a processor at a time. Alternatively, the schedulers on the different systems can communicate, or the application itself can guide the local scheduling process to increase the likelihood that processes will be coscheduled [3], [14].

To summarize the points illustrated by this example: in clusters, the implementation of services taken for granted in end systems can require new approaches to the implementation of existing services (e.g., interprocess communication) and the development of new services (e.g., DSM and coscheduling). The complexity of the new approaches and services, as well as the number of modifications required to the commodity technologies from which clusters are constructed, tends to increase proportionally with performance requirements.

We can paint a similar picture in other areas, such as process creation, process control, and I/O.

Experience shows that familiar services can be extended to the cluster environment without too much difficulty, especially if performance is not critical; the more sophisticated cluster systems provide transparent mechanisms for allocating resources, creating processes, controlling processes, accessing files, and so forth, that work regardless of a program's location within the cluster. However, when performance is critical, new implementation techniques, low-level services, and high-level interfaces can be required [65], [25].

Future Directions

Cluster architectures are evolving in response to three pressures:

1. Performance requirements motivate increased integration and hence operating system and hardware modifications (for example, to support fast communications).

2. Changed operational parameters introduce a need for new operating system and user-level services, such as coscheduling.

3. Economic pressures encourage a continued focus on commodity technologies, at the expense of decreased integration and hence performance and services.

It seems likely that, in the medium term, software architectures for clusters will converge with those for end systems, as end-system architectures address issues of network operation and scale.

5.4 Intranets

The third class of systems that we consider is the intranet, a grid comprising a potentially large number of resources that nevertheless belong to a single organization. Like a cluster, an intranet can assume centralized administrative control and hence a high degree of coordination among resources. The three principal complicating factors that an intranet introduces are as follows:

1. *Heterogeneity:* The end systems and networks used in an intranet are almost certainly of different types and capabilities. We cannot assume a single system image across all end systems.

2. *Separate administration:* Individual systems will be separately administered; this feature introduces additional heterogeneity and the need to negotiate potentially conflicting policies.

3. *Lack of global knowledge:* A consequence of the first two factors, and the increased number of end systems, is that it is not possible, in general, for any one person or computation to have accurate global knowledge of system structure or state.

State of the Art

The software technologies employed in intranets focus primarily on the problems of physical and administrative heterogeneity. The result is typically a simpler, less tightly integrated set of services than in a typical cluster. Commonly, the services that are provided are concerned primarily with the sharing of data (e.g., distributed file systems, databases, Web services) or with providing access to specialized services, rather than with supporting the coordinated use of multiple resources. Access to nonlocal resources often requires the use of simple, high-level interfaces designed for "arm's-length" operation in environments in which every operation may involve authentication, format conversions, error checking, and accounting. Nevertheless, centralized administrative control does mean that a certain degree of uniformity of mechanism and interface can be achieved; for example, all machines

may be required to run a specific distributed file system or batch scheduler, or may be placed behind a firewall, hence simplifying security solutions.

Software architectures commonly used in intranets include the Distributed Computing Environment (DCE), DCOM, and CORBA. In these systems, programs typically do not allocate resources and create processes explicitly, but rather connect to established "services" that encapsulate hardware resources or provide defined computational services. Interactions occur via remote procedure call [33] or remote method invocation [55], [36], models designed for situations in which the parties involved have little knowledge of each other. Communications occur via standardized protocols (typically layered on TCP/IP) that are designed for portability rather than high performance. In larger intranets, particularly those used for mission-critical applications, reliable group communication protocols such as those implemented by ISIS [7] and Totem [46] can be used to deal with failure by ordering the occurrence of events within the system.

The limited centralized control provided by a parent organization can allow the deployment of distributed queuing systems such as Load Sharing Facility (LSF), Codine, or Condor, hence providing uniform access to compute resources. Such systems provide some support for remote management of computation, for example, by distributing a limited range of signals to processes through local servers and a logical signal distribution network. However, issues of security, payment mechanisms, and policy often prevent these solutions from scaling to large intranets.

In a similar fashion, uniform access to data resources can be provided by means of wide area file system technology (such as DFS), distributed database technology, or remote database access (such as SQL servers). High-performance, parallel access to data resources can be provided by more specialized systems such as the High Performance Storage System [73]. In these cases, the interfaces presented to the application would be the same as those provided in the cluster environment.

The greater heterogeneity, scale, and distribution of the intranet environment also introduce the need for services that are not needed in clusters. For example, resource discovery mechanisms may be needed to support the discovery of the name, location, and other characteristics of resources currently available on the network. A reduced level of trust and greater exposure to external threats may motivate the use of more sophisticated security technologies. Here, we can once again exploit the limited centralized control that a parent organization can offer. Solutions such as Kerberos [51] can be mandated and integrated into the computational model, providing a unified authentication structure throughout the intranet.

Future Directions

Existing intranet technologies do a reasonable job of projecting a subset of familiar programming models and services (procedure calls, file systems, etc.) into an environment of greater complexity and physical scale, but are inadequate for performance-driven applications. We expect future developments to overcome these difficulties by extending lighter-weight interaction models originally developed within clusters into the more complex intranet environment, and by developing specialized performance-oriented interfaces to various services.

5.5 Internets

The final class of systems that we consider is also the most challenging on which to perform network computing—internetworked systems that span multiple organizations. Like intranets, internets tend to be large and heterogeneous. The three principal additional complicating factors that an internet introduces are as follows:

1. *Lack of centralized control:* There is no central authority to enforce operational policies or to ensure resource quality, and so we see wide variation in both policy and quality.
2. *Geographical distribution:* Internets typically link resources that are geographically widely distributed. This distribution leads to network performance characteristics significantly different from those in local area or metropolitan area networks of clusters and intranets. Not only does latency scale linearly with distance, but bisection bandwidth arguments [18], [26] suggest that accessible bandwidth tends to decline linearly with distance, as a result of increased competition for long-haul links.
3. *International issues:* If a grid extends across international borders, export controls may constrain the technologies that can be used for security, and so on.

State of the Art

The internet environment's scale and lack of central control have so far prevented the successful widespread deployment of grid services. Approaches that are effective in intranets often break down because of the increased scale and lack of centralized management. The set of assumptions that one user or resource can make about another is reduced yet further, a situation that can lead to a need for implementation techniques based on discovery and negotiation.

We use two examples to show how the internet environment can require new approaches. We first consider security. In an intranet, it can be reasonable to assume that every user has a preestablished trust relationship with every resource that he wishes to access. In the more open internet environment, this assumption becomes intractable because of the sheer number of potential process-to-resource relationships. This problem is accentuated by the dynamic and transient nature of computation, which makes any explicit representation of these relationships infeasible. Free-flowing interaction between computations and resources requires more dynamic approaches to authentication and access control. One potential solution is to introduce the notion of delegation of trust into security relationships: that is, we introduce mechanisms that allow an organization A to trust a user U because user U is trusted by a second organization B, with which A has a formal relationship. However, the development of such mechanisms remains a research problem.

As a second example, we consider the problem of coscheduling. In an intranet, it can be reasonable to assume that all resources run a single scheduler, whether a commercial system such as LSF or a research system such as Condor. Hence, it may be feasible to provide coscheduling facilities in support of applications that need to run on multiple resources at once. In an internet, we cannot rely on the existence of a common scheduling infrastructure. In this environment, coscheduling requires that a grid application (or scheduling service acting for an application) obtain knowledge of the scheduling policies that apply on different resources and influence the schedule either directly through an external scheduling API or indirectly via some other means [16].

Future Directions

Future development of grid technologies for internet environments will involve the development of more sophisticated grid services and the gradual evolution of the services provided at end systems in support of those services. There is little consensus on the shape of the grid architectures that will emerge as a result of this process, but both commercial technologies and research projects point to interesting potential directions. Three of these directions—commodity technologies, Legion, and Globus—are explored in detail in later chapters. We note their key characteristics here but avoid discussion of their relative merits. There is as yet too little experience in their use for such discussion to be meaningful.

The commodity approach to grid architecture adopts as the basis for grid development the vast range of commodity technologies that are emerging at present, driven by the success of the Internet and Web and by the demands of electronic information delivery and commerce. These technologies are being used to construct three-tier architectures, in which middle-tier application servers mediate between sophisticated back-end services and potentially simple front ends. Grid applications are supported in this environment by means of specialized high-performance back-end and application servers.

The Legion approach to grid architecture seeks to use object-oriented design techniques to simplify the definition, deployment, application, and long-term evolution of grid components. Hence, the Legion architecture defines a complete object model that includes abstractions of compute resources called *host objects*, abstractions of storage systems called *data vault objects*, and a variety of other object classes. Users can use inheritance and other object-oriented techniques to specialize the behavior of these objects to their own particular needs, as well as develop new objects.

The Globus approach to grid architecture is based on two assumptions:

1. Grid architectures should provide basic services, but not prescribe particular programming models or higher-level architectures.

2. Grid applications require services beyond those provided by today's commodity technologies.

Hence, the focus is on defining a "toolkit" of low-level services for security, communication, resource location, resource allocation, process management, and data access. These services are then used to implement higher-level services, tools, and programming models.

In addition, hybrids of these different architectural approaches are possible and will almost certainly be addressed: for example, a commodity three-tier system might use Globus services for its back end.

A wide range of other projects are exploring technologies of potential relevance to computational grids, for example, WebOS [67], Charlotte [6], UFO [2], ATLAS [5], Javelin [15], Popcorn [10], and Globe [69].

6 Research Challenges

What problems must be solved to enable grid development? In preceding sections, we outlined what we expect grids to look like and how we expect them to be used. In doing so, we tried to be as concrete as possible, with the goal of providing at least a plausible view of the future. However, there are certainly many challenges to be overcome before grids can be used as easily and flexibly as we have described. In this section, we summarize the nature of these challenges, most of which are discussed in much greater detail in the chapters that follow.

6.1 The Nature of Applications

Early metacomputing experiments provide useful clues regarding the nature of the applications that will motivate and drive early grid development. However, history also tells us that dramatic changes in capabilities such as those discussed here are likely to lead to radically new ways of using computers—ways as yet unimagined. Research is required to explore the bounds of what is possible, both within those scientific and engineering domains in which metacomputing has traditionally been applied, and in other areas such as business, art, and entertainment.

6.2 Programming Models and Tools

As noted in Section 4, grid environments will require a rethinking of existing programming models and, most likely, new thinking about novel models more suitable for the specific characteristics of grid applications and environments. Within individual applications, new techniques are required for expressing advanced algorithms, for mapping those algorithms onto complex grid architectures, for translating user performance requirements into system resource requirements, and for adapting to changes in underlying system structure and state. Increased application and system complexity increases the importance of code reuse, and so techniques for the construction and composition of grid-enabled software components will be important. Another significant challenge is to provide tools that allow programmers to understand and explain program behavior and performance.

6.3 System Architecture

The software systems that support grid applications must satisfy a variety of potentially conflicting requirements. A need for broad deployment implies that these systems must be simple and place minimal demands on local sites. At the same time, the need to achieve a wide variety of complex, performance-sensitive applications implies that these systems must provide a range of potentially sophisticated services. Other complicating factors include the need for scalability and evolution to future systems and services. It seems likely that new approaches to software architecture will be needed to meet these requirements—approaches that do not appear to be satisfied by existing Internet, distributed computing, or parallel computing technologies.

6.4 Algorithms and Problem-Solving Methods

Grid environments differ substantially from conventional uniprocessor and parallel computing systems in their performance, cost, reliability, and security characteristics. These new characteristics will undoubtedly motivate the development of new classes of problem-solving methods and algorithms. Latency-tolerant and fault-tolerant solution strategies represent one important area in which research is required [5], [6], [10]. Highly concurrent and speculative execution techniques may be appropriate in environments where many more resources are available than at present.

6.5 Resource Management

A defining feature of computational grids is that they involve sharing of networks, computers, and other resources. This sharing introduces challenging resource management problems that are beyond the state of the art in a variety of areas. Many of the applications described in later chapters need to meet stringent end-to-end performance requirements across multiple computational resources connected by heterogeneous, shared networks. To meet these requirements, we must provide improved methods for specifying application-level requirements, for translating these requirements into computational resources and network-level quality-of-service parameters, and for arbitrating between conflicting demands.

6.6 Security

Sharing also introduces challenging security problems. Traditional network security research has focused primarily on two-party client-server interactions with relatively low performance requirements. Grid applications frequently involve many more entities, impose stringent performance requirements, and involve more complex activities such as collective operations and the downloading of code. In larger grids, issues that arise in electronic markets become important. Users may require assurance

and licensing mechanisms that can provide guarantees (backed by financial obligations) that services behave as advertised [37].

6.7 Instrumentation and Performance Analysis

The complexity of grid environments and the performance complexity of many grid applications make techniques for collecting, analyzing, and explaining performance data of critical importance. Depending on the application and computing environment, poor performance as perceived by a user can be due to any one or a combination of many factors: an inappropriate algorithm, poor load balancing, inappropriate choice of communication protocol, contention for resources, or a faulty router. Significant advances in instrumentation, measurement, and analysis are required if we are to be able to relate subtle performance problems in the complex environments of future grids to appropriate application and system characteristics.

6.8 End Systems

Grids also have implications for the end systems from which they are constructed. Today's end systems are relatively small and are connected to networks by interfaces and with operating system mechanisms originally developed for reading and writing slow disks. Grids require that this model evolve in two dimensions. First, by increasing demand for high-performance networking, grid systems will motivate new approaches to operating system and network interface design in which networks are integrated with computers and operating systems at a more fundamental level than is the case today. Second, by developing new applications for networked computers, grids will accelerate local integration and hence increase the size and complexity of the end systems from which they are constructed.

6.9 Network Protocols and Infrastructure

Grid applications can be expected to have significant implications for future network protocols and hardware technologies. Mainstream developments in networking, particularly in the Internet community, have focused on best-effort service for large numbers of relatively low-bandwidth flows. Many of the future grid applications discussed in this book require both high bandwidths and stringent performance assurances. Meeting these requirements will require major advances in the technologies used to transport, switch, route, and manage network flows.

7 Summary

This chapter has provided a high-level view of the expected purpose, shape, and architecture of future grid systems and, in the process, sketched a road map for more detailed technical discussion in subsequent chapters. The discussion was structured in terms of six questions.

Why do we need computational grids? We explained how grids can enhance human creativity by, for example, increasing the aggregate and peak computational performance available to important applications and allowing the coupling of geographically separated people and computers to support collaborative engineering. We also discussed how such applications motivate our requirement for a software and hardware infrastructure able to provide dependable, consistent, and pervasive access to high-end computational capabilities.

What types of applications will grids be used for? We described five classes of grid applications: distributed supercomputing, in which many grid resources are used to solve very large problems; high throughput, in which grid resources are used to solve large numbers of small tasks; on demand, in which grids are used to meet peak needs for computational resources; data intensive, in which the

focus is on coupling distributed data resources; and collaborative, in which grids are used to connect people.

Who will use grids? We examined the shape and concerns of four grid communities, each supporting a different type of grid: a national grid, serving a national government; a private grid, serving a health maintenance organization; a virtual grid, serving a scientific collaboratory; and a public grid, supporting a market for computational services.

How will grids be used? We analyzed the requirements of five classes of users for grid tools and services, distinguishing between the needs and concerns of end users, application developers, tool developers, grid developers, and system managers.

What is involved in building a grid? We discussed potential approaches to grid architecture, distinguishing between the differing concerns that arise and technologies that have been developed within individual end systems, clusters, intranets, and internets.

What problems must be solved to enable grid development? We provided a brief review of the research challenges that remain to be addressed before grids can be constructed and used on a large scale.

Further Reading

For more information on the topics covered in this chapter, see *www.mkp.com/grids* and also the following references:

- A series of books published by the Corporation for National Research Initiatives [29], [30], [31], [28] review and draw lessons from other large-scale infrastructures, such as the electric power grid, telecommunications network, and banking system.
- Catlett and Smarr's original paper on metacomputing [13] provides an early vision of how high-performance distributed computing can change the way in which scientists and engineers use computing.
- Papers in a 1996 special issue of the *International Journal of Supercomputer Applications* [19] describe the architecture and selected applications of the I-WAY metacomputing experiment.
- Papers in a 1997 special issue of the *Communications of the ACM* [62] describe plans for a National Technology Grid.
- Several reports by the National Research Council touch upon issues relevant to grids [49], [50], [48].
- Birman and van Renesse [8] discuss the challenges that we face in achieving reliability in grid applications.

References

[1] D. Abramson, R. Sosic, J. Giddy, and B. Hall. Nimrod: A tool for performing parameterised simulations using distributed workstations. In *Proc. 4th IEEE Symp. on High Performance Distributed Computing*. IEEE Computer Society Press, 1995.

[2] A. D. Alexandrov, M. Ibel, K. E. Schauser, and C. J. Scheiman. Extending the operating system at the user level: The UFO global file system. In *1997 Annual Technical Conference on UNIX and Advanced Computing Systems (USENIX'97)*. January 1997.

[3] T. Anderson. Glunix: A global layer Unix for NOW. http://now.cs.berkeley.edu/Glunix/glunix.html.

[4] R. Arpaci, A. Dusseau, A. Vahdat, L. Liu, T. Anderson, and D. Patterson. The interaction of parallel and sequential workloads on a network of workstations. In *Proc. SIGMETRICS*, 1995.

[5] J. Baldeschwieler, R. Blumofe, and E. Brewer. ATLAS: An infrastructure for global computing. In *Proc. Seventh ACM SIGOPS European Workshop on System Support for Worldwide Applications*, 1996.

[6] A. Baratloo, M. Karaul, Z. Kedem, and P. Wyckoff. Charlotte: Metacomputing on the Web. In *Proc. 9th Conference on Parallel and Distributed Computing Systems*, 1996.

[7] K. P. Birman and R. van Rennesse. *Reliable Distributed Computing Using the Isis Toolkit*. IEEE Computer Society Press, 1994.

[8] Kenneth P. Birman and Robbert van Renesse. Software for reliable networks. *Scientific American*, May 1996.

[9] Richard Bukowski and Carlo Sequin. Interactive simulation of fire in virtual building environments. In *Proceedings of SIGGRAPH 97*, 1997.

[10] N. Camiel, S. London, N. Nisan, and O. Regev. The POPCORN project: Distributed computation over the Internet in Java. In *Proc. 6th International World Wide Web Conference*, 1997.

[11] Henri Casanova and Jack Dongarra. Netsolve: A network server for solving computational science problems. Technical Report CS-95-313, University of Tennessee, November 1995.

[12] J. Casazza. *The Development of Electric Power Transmission: The Role Played by Technology, Institutions and People*. IEEE Computer Society Press, 1993.

[13] C. Catlett and L. Smarr. Metacomputing. *Communications of the ACM*, 35(6):44–52, 1992.

[14] A. Chien, S. Pakin, M. Lauria, M. Buchanan, K. Hane, L. Giannini, and J. Prusakova. High performance virtual machines (HPVM): Clusters with supercomputing APIs and performance. In *Eighth SIAM Conference on Parallel Processing for Scientific Computing (PP97)*, March 1997.

[15] B. Christiansen, P. Cappello, M. Ionescu, M. Neary, K. Schauser, and D. Wu. Javelin: Internet-based parallel computing using Java. In *Proc. 1997 Workshop on Java in Computational Science and Engineering*, 1997.

[16] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. In *The 4th Workshop on Job Scheduling Strategies for Parallel Processing*, 1998.

[17] Joseph Czyzyk, Michael P. Mesnier, and Jorge J. Moré. The Network-Enabled Optimization System (NEOS) Server. Preprint MCS-P615-0996, Argonne National Laboratory, Argonne, Illinois, 1996.

[18] W. Dally. *A VLSI Architecture for Concurrent Data Structures*. Kluwer Academic Publishers, 1987.

[19] T. DeFanti, I. Foster, M. Papka, R. Stevens, and T. Kuhfuss. Overview of the i-way: Wide area visual supercomputing. *International Journal of Supercomputer Applications*, 10(2):123–130, 1996.

[20] D. Diachin, L. Freitag, D. Heath, J. Herzog, W. Michels, and P. Plassmann. Remote engineering tools for the design of pollution control systems for commercial boilers. *International Journal of Supercomputer Applications*, 10(2):208–218, 1996.

[21] F. Douglis and J. Ousterhout. Transparent process migration: Design alternatives and the Sprite implementation. *Software—Practice and Experience*, 21(8):757–85, 1991.

[22] Peter Druschel, Mark B. Abbott, Michael A. Pagels, and Larry L. Peterson. Network subsystem design. *IEEE Network*, 7(4):8–17, July 1993.

[23] Andrea C. Dusseau, Remzi H. Arpaci, and David E. Culler. Effective distributed scheduling of parallel workloads. In *ACM SIGMETRICS '96 Conference on the Measurement and Modeling of Computer Systems*, 1996.

[24] S. Dwarkadas, P. Keleher, A. Cox, and W. Zwaenepoel. An evaluation of software distributed shared memory for next-generation processors and networks. In *Proceedings of the 20th International Symposium on Computer Architecture*, San Diego, CA, May 1993.

[25] D. Engler, M. Kaashoek, , and J. O'Toole Jr. Exokernel: An operating system architecture for application-level resource management. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pages 251–266. ACM Press, 1995.

[26] I. Foster. *Designing and Building Parallel Programs*. Addison-Wesley, 1995.

[27] I. Foster, J. Geisler, W. Gropp, N. Karonis, E. Lusk, G. Thiruvathukal, and S. Tuecke. A wide-area implementation of the Message Passing Interface. *Parallel Computing*, 1998. to appear.

[28] Amy Friedlander. *In God We Trust All Others Pay Cash: Banking as an American Infrastructure 1800–1935*. Corporation for National Research Initiatives, Reston, VA, 199.

[29] Amy Friedlander. *Emerging Infrastructure: The Growth of Railroads*. Corporation for National Research Initiatives, Reston, VA, 1995.

[30] Amy Friedlander. *Natural Monopoly and Universal Service: Telephones and Telegraphs in the U.S. Telecommunications Infrastructure 1837–1940*. Corporation for National Research Initiatives, Reston, VA, 1995.

[31] Amy Friedlander. *Power and Light: Electricity in the U.S. Energy Infrastructure 1870–1940*. Corporation for National Research Initiatives, Reston, VA, 1996.

[32] I. Goldberg, D. Wagner, R. Thomas, and E. Brewer. A secure environment for untrusted helper applications. In *Proceedings of the Sixth Usenix Security Symposium*, July 1996.

[33] Jr Harold Lockhart. *OSF DCE: Guide to Developing Distributed Applications*. McGraw Hill, 1994.

[34] Bernardo Huberman, editor. *The Ecology of Computation*. Elsevier Science Publishers/North-Holland, 1988.

[35] Van Jacobson. Efficient protocol implementation. In *ACM SIGCOMM '90 tutorial*, September 1990.

[36] JavaSoft. RMI. The JDK 1.1 Specification. http://javasoft.com/products/jdk/1.1/docs/guide/rmi/index.html, 1997.

[37] Charlie Lai, Gennady Medvinsky, and Clifford Neuman. Endorsements, licensing, and insurance for distributed system services. In *Proceedings of the Second ACM Conference on Computer and Communications Security*, November 1994.

[38] C. Lee, C. Kesselman, and S. Schwab. Near-realtime satellite image processing: Metacomputing in CC++. *IEEE Computer Graphics and Applications*, 16(4):79–84, 1996.

[39] Jason Leigh, Andrew Johnson, and Thomas A. DeFanti. CAVERN: A distributed architecture for supporting scalable persistence and interoperability in collaborative virtual environments. *Virtual Reality: Research, Development and Applications*, 2(2):217–237, December 1997.

[40] A. Lenstra. Factoring integers using the Web and the number field sieve. Technical report, Bellcore, August 1995.

[41] Michael J. Litzkow, Miron Livny, and Matt W. Mutka. Condor—a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, pages 104–111, June 1988.

[42] P. Lyster, L. Bergman, P. Li, D. Stanfill, B. Crippe, R. Blom, C. Pardo, and D. Okaya. CASA gigabit supercomputing network: CALCRUST three-dimensional real-time multi-dataset rendering. In *Proc. Supercomputing '92*, 1992.

[43] K. Marzullo, M. Ogg, A. Ricciardi, A. Amoroso, F. Calkins, and E. Rothfus. NILE: Wide-area computing for high energy physics. *Proceedings of the 1996 SIGOPS Conference*, 1996.

[44] G. McGraw and E. Felten. *Java Security: Hostile Applets, Holes and Antidotes*. John Wiley and Sons, 1996.

[45] C. Mechoso. C.-C. Ma. J. Farrara. J. Spahr. and R. Moore. Parallelization and distribution of a coupled atmosphere-ocean general circulation model. *Mon. Wea. Rev.*. 121:2062. 1993.

[46] L. E. Moser. P. M. Melliar-Smith. D. A. Agarwal. R. K. Budhia. and C. A. Lingley-Papadopoulos. Totem: A fault-tolerant multicast group communication system. *Communications of the ACM*. 39(4):54–63. April 1996.

[47] M. Mutka and M. Livny. The available capacity of a privately owned workstation environment. *Performance Evaluation*. 12(4):269–84. 1991.

[48] National Research Council. *National Collaboratories: Applying Information Technology for Scientific Research*. National Academy Press. 1993.

[49] National Research Council. *Evolving the High Performance Computing and Communications Initiative to Support the Nation's Information Infrastructure*. National Academy Press. 1995.

[50] National Research Council. *More Than Screen Deep: Toward Every-Citizen Interfaces to the Nation's Information Infrastructure*. National Academy Press. 1997.

[51] B. Clifford Neuman and Theodore Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications*. 32(9). September 1994.

[52] J. Nieplocha and R. Harrison. Shared memory NUMA programming on the I-WAY. In *Proc. 5th IEEE Symp. on High Performance Distributed Computing*. pages 432–441. IEEE Computer Society Press. 1996.

[53] J. Nieplocha. R.J. Harrison. and R.J. Littlefield. Global Arrays: A portable "shared-memory" programming model for distributed memory computers. In *Proceedings of Supercomputing '94*. pages 340–349. IEEE Computer Society Press. 1994.

[54] M. Norman. P. Beckman. G. Bryan. J. Dubinski. D. Gannon. L. Hernquist. K. Keahey. J. Ostriker. J. Shalf. J. Welling. and S. Yang. Galaxies collide on the I-WAY: An example of heterogeneous wide-area collaborative supercomputing. *International Journal of Supercomputer Applications*. 10(2):131–140. 1996.

[55] Object Management Group. Inc.. Framingham. MA. *The Common Object Request Broker Architecture and Specifications*. version 2.0 edition. July 1996.

[56] Scott Pakin. Vijay Karamcheti. and Andrew A. Chien. Fast Messages: Efficient. portable communication for workstation clusters and mpps. *IEEE Concurrency*. 5(2):60–73. April-June 1997.

[57] C. Potter. R. Brady. P. Moran. C. Gregory. B. Carragher. N. Kisseberth. J. Lyding. and J. Lindquist. EVAC: A virtual environment for control of remote imaging instrumentation. *IEEE Computer Graphics and Applications*. pages 62–66. 1996.

[58] C. Potter. Z-P. Liang. C. Gregory. H. Morris. and P. Lauterbur. Toward a neuroscope: A real-time system for the evaluation of brain function. In *Proc. First IEEE Int'l Conf. on Image Processing*. volume 3. pages 25–29. IEEE Computer Society Press. 1994.

[59] I Richer and B Fuller. The MAGIC project: From vision to reality. *IEEE Network*. May/June 1996.

[60] Maria Roussos. Andrew Johnson. Jason Leigh. Christina Valsilakis. Craig Barnes. and Thomas Moher. NICE: Combining constructionism. narrative. and collaboration in a virtual learning environment. *Computer Graphics*. 31(3):62–63. August 1997.

[61] A. Silberschatz. J. Peterson. and P. Galvin. *Operating Systems Concepts*. Addison-Wesley. 1991.

[62] Larry Smarr. Computational infrastructure: Toward the 21st century. *Communications of the ACM*. 40(11). November 1997.

[63] Patrick G. Sobalvarro and William E. Weihl. Demand-based coscheduling of parallel jobs on multiprogrammed multiprocessors. In *Proceedings of the Parallel Job Scheduling Workshop at IPPS '95*. 1995.

[64] W. Sullivan. D. Werthimer. S. Bowyer. J. Cobb. D. Gedye. and D. Anderson. A new major SETI project based on project SERENDIP data and 100,000 personal computers. In *Astronomical and Biochemical Origins and the Search for the Life in the Universe*. 1997. IAU Colloquium No. 161.

[65] R. Unrau, O. Krieger, B. Gamsa, and M. Stumm. Hierarchical clustering: A structure for scalable multiprocessor operating system design. *The Journal of Supercomputing*, 9(1/2):105–134. 1995.

[66] A. Vahdat, P. Eastham, and T. Anderson. WebFS: A global cache coherent filesystem. Technical report, Department of Computer Science, UC Berkeley, 1996.

[67] A. Vahdat, P. Eastham, C. Yoshikawa, E. Belani, T. Anderson, D. Culler, and M. Dahlin. WebOS: Operating system services for wide area applications. Technical Report UCB CSD-97-938, U.C. Berkeley, 1997.

[68] R. van Renesse, K. P. Birman, and S. Maffeis. Horus: A flexible group communication system. *Communications of the ACM*, 39(4):76–83, April 1996.

[69] M. van Steen, P. Homburg, L. van Doorn, A. Tanenbaum, and W. de Jonge. Towards object-based wide area distributed systems. In *Proc. International Workshop on Object Orientation in Operating Systems*, pages 224–227, 1995.

[70] T. von Eicken, D. Culler, S. Goldstein, and K. Schauser. Active messages: A mechanism for integrated communication and computation. In *Proceedings of the 19th International Symposium on Computer Architecture*, pages 256–266. ACM Press, May 1992.

[71] R. Wahbe, S. Lucco, T. Anderson, and S. Graham. Efficient software-based fault isolation. In *Proc. 14th Symposium on Operating System Principles*, 1993.

[72] D. Wallach, D. Balfanz, D. Dean, and E. Felten. Extensible security in Java. Technical Report 546-97, Dept of Computer Science, Princeton University, 1997.

[73] R. Watson and R. Coyne. The parallel I/O architecture of the high performance storage system (HPSS). In *14th IEEE Symposium Mass Storage Systems*, Monterey, CA, September 1995. Comp. Soc. Press.

[74] Glen H. Wheless, Cathy M. Lascara, Arnoldo Valle-Levinson, Donald P. Brutzman, William Sherman, William L. Hibbard, and Brian E. Paul. Virtual chesapeake bay: Interacting with a coupled physical/biological model. *IEEE Computer Graphics and Applications*, 16(4):42–43, July 1996.

[75] S. Zhou. LSF: Load sharing in large-scale heterogeneous distributed systems. In *Proc. Workshop on Cluster Computing*, 1992.

[76] S. Zhou, M. Stumm, K. Li, and D. Wortmann. Heterogeneous distributed shared memory (Mermaid). *IEEE Transactions on Parallel and Distributed Systems*, 3(5):540–554, September 1992.

Implementing and Analysing an Effective Explicit Coscheduling Algorithm on a NOW*

Francesc Solsona[1], Francesc Giné[1], Fermin Molina[1], Porfidio Hernández[2], and Emilio Luque[2]

[1] University of LLeida, Dept. of CS,
Jaume II 69, 25001 LLeida, Spain.
francesc,sisco,fermin@eup.udl.es
[2] UAB, Dept. of CS,
08193 Bellaterra, Barcelona, Spain.
p.hernandez,e.luque@cc.uab.es

Abstract. Networks of workstations (NOWs) have become important and cost-effective parallel platforms for scientific computations. In practice, a NOW system is heterogeneous and non-dedicated. These two unique factors make scheduling policies on multiprocessor/multicomputer systems unsuitable for NOWs, but the coscheduling principle is still an important basis for parallel process scheduling in these environments. The main idea of this technique is to schedule the set of tasks composing a parallel application at the same time, to increase their communication performance. In this article we present an explicit coscheduling algorithm implemented in a Linux NOW, of PVM distributed tasks, based on Real Time priority assignment. The main goal of the algorithm is to execute efficiently distributed applications without excessively damaging the response time of local tasks. Extensive performance analysis as well as studies of the parameters and overheads involved in the implementation demonstrated the applicability of the proposed algorithm.

1 Introduction

Parallel and distributed computing in a network of workstations (NOWs) receives ever increasing attention. Recently, a research goal is to build a NOW that runs parallel programs with performance equivalent to a MPP (Massively Parallel Processor) and executes sequential programs as a dedicated uniprocessor too. Nevertheless, two issues must be addressed: how to coordinate the simultaneous execution of the processes of a parallel job, and how to manage the interaction between parallel and local user jobs.

The studies in [1] indicate that the workstations in a NOW are normally underloaded. Basically, there are two methods of making use of these CPU idle cycles, task migration [2, 3] and job scheduling [4–6]. In a NOW, in accordance with the research realized by Arpaci [7], task migration overheads and the unpredictable behavior of local users may lower the effectiveness of this method.

* This work was supported by the CICYT under contract TIC98-0433

Our research was focussed on the approach of keeping both local and parallel jobs together and effective, and efficiently scheduling them.

A NOW system is heterogeneous and non-dedicated. The heterogeneity can be modeled by the Power weight [8]. As for the non-dedicated feature, a mechanism must be provided to ensure that no extra context switch overheads due to synchronization delays are introduced. Outerhout's solution for timeshared multiprocessor systems was coscheduling [9]. Under this traditional form of coscheduling, the processes constituting a parallel job are scheduled simultaneously across as many nodes of a multiprocessor as they require.

Explicit coscheduling [9, 5] ensures that scheduling of communicating jobs is coordinated by constructing a static global list of the order in which jobs should be scheduled; a simultaneous global context switch is then required in all the processors. Zhang [4], based on the coscheduling principle, has implemented the so-called "self-coordinated local scheduler", which guarantees the performance of both local and parallel jobs in a NOW by a time-sharing and priority-based operating system. He varies the priority of the processes according to the power usage agreement between local and parallel jobs.

In contrast with Zhang's study, a real implementation of explicit coscheduling in a NOW is presented in this article; so that the user of the parallel machine has all the computing power of the NOW available during a short period of time with the main aim of obtaining good performance of distributed tasks without excessively damaging the local ones.

In section 2, the environment DTS (Distributed Scheduler) where the coscheduling implementation is built will be introduced. In section 3, our explicit coscheduling algorithm of PVM distributed tasks in a Linux NOW is presented. Also, a synchronization algorithm that improves the performance of the message passing in distributed tasks is proposed. In section 4. the good behavior of the implemented algorithms is checked by means of measuring the execution time on both synthetic applications and NAS benchmarks: in addition, the response time of local jobs and other parameters and overheads of special interest are measured. Finally, the last section includes our conclusions and a description of the future work.

2 DTS Environment

We are interested in assigning a period of time to distributed tasks and another to interactive ones, and varying these dynamically according to the local load average of a NOW. Also, our aim is to avoid modifying the kernel source, because of the need of a portable system. Our solution consists of promoting the distributed tasks (initially timesharing) to real-time. Furthermore, in each workstation all the distributed tasks were put in the same group of processes: it allows control of their execution by means of stop and resume signals. In such a way that, this splits the CPU time in two different periods, the parallel slice and the interactive one.

The implemented system, called DTS, which is an improved version of the original described in [10] is composed of three types of modules, the *Scheduler*, the *Load* and the *Console*. The *Scheduler* and the *Load* are composed of distributed processes running on each active workstation. The function of each Scheduler is the dynamic variation of the amount of CPU cycles exclusively assigned to execute distributed tasks (*PS: Parallel Slice*), and the amount of time assigned to local tasks (*IS: Interactive Slice*). The *Load* processes collect the interactive load on every workstation. The *Console* can be executed from any node of the NOW and is responsible for managing and controlling the system. For notation convenience the set of active nodes in the NOW are called VM (Virtual Machine), (see Figure 1).

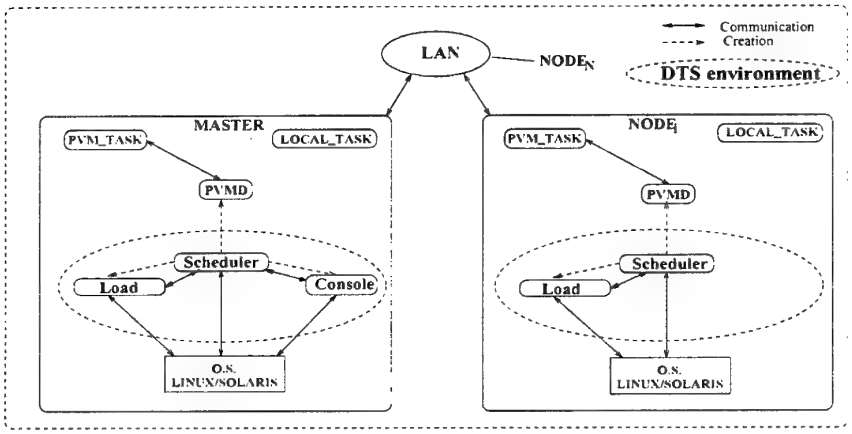


Fig. 1. DTS environment.

Our environment is started running automatically by the pvm environment. In each workstation composing our VM, the pvmd shell script has been modified as follows: the sentence "*exec $PVM_ROOT/lib/pvmd3 $@*" has been changed to "*exec $PVM_ROOT/lib/scheduler $@*". This way, when the workstation is added/activated to the virtual machine (even if it is the first) from the pvm console, the *Scheduler* is executed.

3 Coscheduling Implementation

In this section the coscheduling algorithm implemented over the DTS's scheduler daemon is explained. Furthermore, some improvements in communication performance of the system are presented with the addition of a synchronization algorithm of the distributed tasks.

3.1 Coscheduling Algorithm

The coscheduling algorithm is shown in the Figure 2.

Scheduler

```
        set PRI(Scheduler) = ((max(rt_priority)) and SCHED_FIFO)
        fork&exec(Load)
        fork&exec(pvmd)
        set PRI(pvmd) = ((max(rt_priority) - 1) and SCHED_RR)
        set PRI(Load) = ((max(rt_priority) - 2) and SCHED_FIFO)
        set pvmd leader of pvm_tasks
sync_p:
        while(pvm_tasks) do
            sleep(PS)
            signal_stop(pvm_tasks)
            sleep(IS)
            signal_resume(pvm_tasks)
        end/*while*/
```

Fig. 2. Coscheduling algorithm

At the beginning of execution, the *Scheduler*, which has root permissions, promotes itself to Real Time class (initially time shared). After that, it forks and executes *Load* and *pvmd* (the pvm daemon), and also promotes *pvmd* and *Load* to -1 and -2 Real Time priority lower than *Scheduler* respectively. Next, *Scheduler*, sets *pvmd* to become the leader of a new group of processes (denoted by *pvm_tasks*; this group will be composed of all the pvm tasks that *pvmd* will create).

The scheduling policy of every process (SCHED_FIFO or SCHED_RR) is shown in the algorithm too, and denotes a FIFO or Round Robin scheduling respectively. *Scheduler* and *Load* have a FIFO policy because of their need to finish their work completely before they release the CPU. On the other hand, *pvmd* can block waiting for the receipt of an event, and meanwhile grant the CPU to another process, perhaps at the same priority level (a pvm task). For this reason, the scheduling policy has to be Round Robin.

Following this, the Scheduler enters in a loop where each iteration takes IP (Iteration Period) ms, where $IP = PS + IS$. This loop stops when there are no more pvm tasks (including the pvmd). This occurs when the workstation is deleted from the pvm console.

Thus, after the Parallel Slice (PS), all the Schedulers in the VM stop all the pvm tasks by sending a STOP signal to the group of PVM processes leadered by pvmd. After that, they are resumed at the end of the interactive slice by sending them a CONTINUE signal. Because the distributed tasks are running in real-time class, they have a global priority higher than the interactive ones.

Thus, when they are resumed, they take control of the CPU. In this way, an interval of execution is assigned to distributed tasks and other interval (can be different) to interactive ones. Figure 3 shows the behavior of the DTS scheduler: each time the Scheduler is executed (during an insignificant time), it concedes the CPU alternatively to the distributed (PS period) and to the interactive (IS period) tasks.

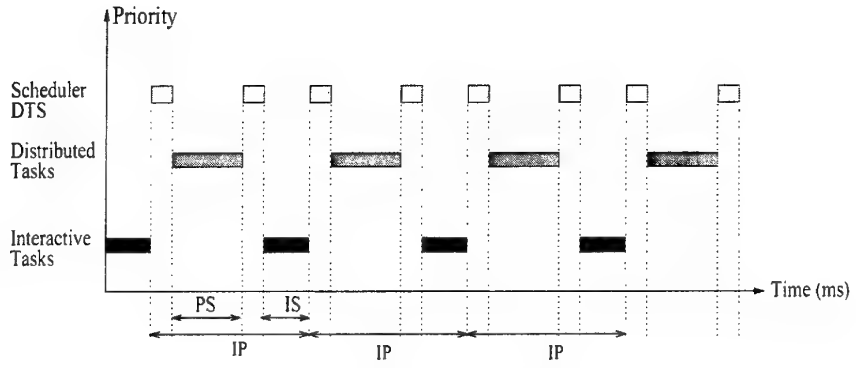


Fig. 3. DTS environment behavior

3.2 Scheduler Synchronization

Only the algorithm of each Scheduler that runs in the VM has been explained, but how are they synchronized to execute the parallel and the interactive slice at the same time and how are these slices modified according to the Load Average in the VM? Figure 4 shows the schematic algorithm that has been used to solve these two questions.

The tasks composing a distributed application can have basically CPU or message passing intensive phases. In the first case, it is unnecessary to synchronize the tasks. On the other hand, in the second case, the synchronization between the communication tasks can increase the global performance of the distributed application [7]. For this reason, DTS has two different modes of operation. In the Dynamic mode, the PS and IS periods are synchronized over all the distributed tasks, whereas in the Distributed mode, the CPU intensive tasks are not synchronized at all.

Every Load Interval (LI), all the *Load* processes collect the real CPU queue length (q_i). The work done by Ferrary [11] shows that the length of the ready queue is a good index for measuring the load of a workstation. After N (Number of LI intervals of passed history to take into account) the *Load Index*, denoted as Q_i, is computed and a message containing the load is sent to the *Console*. Exponential smoothing is used to compute the *Load Index*, defined as follows:

```
Load_j: ∀M_j ∈ VM
        i = 0, Q^j_{i-1} = 0
        Each LI interval do
              collect(q_i)
              collect(Net_Activity)
              compute(Q^j_i)
              if (++i mod(N) == 0)
                 if (Net_Activity ≤ Network_Treshold)
                    set MODE_DTS = DISTRIBUTED
                    compute(PS&IS)
                    set PS&IS
                 else
                    set MODE_DTS = DYNAMIC
                    if (|Q^j_i − Q^j_{i-1}| ≤ Load_Treshold))
                       send(Console.Q^j_i)

Console: Node Master
         if (MODE_DTS==DYNAMIC)
            while(timeout)do
               for each M_j ∈ VM async_receive(Q^j_i)
            compute(RLA,PS&IS)
            broadcast(PS&IS)

Scheduler_j: ∀M_j ∈ VM
            if (MODE_DTS == DYNAMIC)
               async_receive(PS&IS)
               set PS&IS
               goto sync_p
```

Fig. 4. Synchronization Algorithm. *Console* is in the node master. There is a *Load* and *Scheduler* module in each node of the VM

$$Q_i = Q_{i-1}e^{-P} + q_i(1 - e^{-P}), i \geq 1, \quad (1)$$
$$Q_0 = 0,$$

where Q_{i-1} is the last computed index, q_i is the real CPU queue and $P = \frac{1}{N}$. Taking into account the studies done by Ferrari [11], a LI of 100 ms and N of 10 has been chosen.

Note that when *Load* collects q_i, the distributed tasks are stopped, waiting out of the Ready queue. For this reason, the distributed tasks are not computed. In another situation, as for example systems where the priority of distributed tasks is increased and decreased periodically, the need to distinguish between distributed and interactive tasks adds a great overhead to the system.

In Figure 4 is important to observe how DTS activates automatically either the DISTRIBUTED or the DYNAMIC mode of operation in each workstation.

depending on the network activity (*Net_Activity*). The network activity is the number of messages sent or received every $N * LI$ intervals by the *pvmd*. The behavior of the nodes which operates in DISTRIBUTED or DYNAMIC mode is explained separately below.

One centralized algorithm has been implemented, due basically to performance requirements of the local applications. On the other hand, if the algorithm was distributed, it would reduce the performance of the interactive tasks and would increase the network activity due to the high activity of the *Load* module, for example sending the load index of each node to all the VM.

Dynamic Mode In the reception of the Load indexes from all the active nodes or after a timeout, the *Console* computes the *Relative Load Average* (RLA), a metric used in DYNAMIC mode to fix the parallel and interactive slice on each workstation. The RLA is defined as follows:

$$RLA = \frac{\sum_{j=1}^{NW} \frac{Q_i^j}{W_j(A)}}{NW} \tag{2}$$

where Q_i^j is the *load index* of workstation j, NW the number of workstations in the VM and $W_j(A)$ the *power weight* of workstation j. The *power weight* [8] is defined as follows:

$$W_j(A) = \frac{S_j(A)}{max_{k=1}^{NW}\{S_k(A)\}}, j = 1...NW \tag{3}$$

where $S_j(A)$ is the speed of the workstation M_j to solve an application of size A on a dedicated system. Nevertheless, the experimental results in [8] show that if applications fit in the main memory, the power weight differences, using several applications, are insignificant.

Table 1, which shows the relation between the RLA, PS and IS, is used to compute the PS and IS. The values of PS and IS shown in the table are percentages of the IP period.

Table 1. Relation between RLA, PS and IS

RLA	IS	PS
0≤ RLA ≤0.25	10	90
0.25< RLA ≤0.5	20	80
0.5< RLA ≤0.75	30	70
0.75< RLA ≤1	40	60
1< RLA ≤1.25	50	50
1.25< RLA ≤1.5	60	40
1.5< RLA ≤1.75	70	30
1.75< RLA ≤2	80	20
2< RLA	90	10

The *Console* sends PS and IS to all the *Schedulers* modules by a broadcast message. Broadcast delivery has been chosen due to the high cost of multicasting or sending a message separately to each node of the VM. On asynchronous reception (done by a message handler), each *Scheduler* process sets its parallel and interactive slices and jumps to a predetermined address, the label sync_p (synchronization point) in the algorithm of Figure 2.

Distributed Mode Each workstation executing in this mode does not exchange many messages, the communication can even be null. Therefore, synchronization is not required. The only factor to take into account is the efficient share of the CPU between the distributed and local tasks, so that each workstation sets the PS and IS slices according to its own sequential workload. Each node computes these values according to Table 1 too. but substituting RLA for Q_i (the Load Index).

4 Experimentation

Our experimental environment is composed of eight 350 MHz Pentium with 128 MB of memory and 512 KB of cache. All of them are connected through an Ethernet network of 100Mbps bandwidth and a minimal latency in the order of 0.1 ms. All our parallel experimentation was carried out in a non dedicated environment with an owner workload (defined as the averaged ready queue length) of 0.5 (Light), 1 (Medium) and 2 (Heavy). Workload characterization was carried out by means of running a variable number of processes in background, representatives of the typical programs of personal workstations. The performance of the coscheduling implementation was evaluated by running three kernel benchmarks from the NAS parallel benchmarks suite [12]: *ep*, an embarrassingly application, *is*. an integer sorting parallel application and *mg*. a parallel application that solves the Poisson problem using multi-grid iterations. Also. two synthetic applications. *sinring* and *sintree* were implemented, representative of two types of communication patterns. The first implements a logical ring (see Figure 5(a)), and the second attends for the communication of one to vary, and vary to one (see Figure 5(b)). In both applications, every node executes different processes sequentially during a fixed period of time, which is an input argument (1 ms by default). The number of iterations of both problems is also an input argument. Table 2 shows the benchmarks parameters used in the experimentation and the execution times obtained with PVM.

Two different performance indexes are used:

- Gain (G): This metric was used for evaluating the performance of our DTS system with respect to the original PVM. The gain is defined as follows:

$$G = \frac{T_{pvm}}{T_{sched}} \tag{4}$$

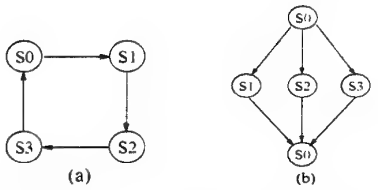


Fig. 5. benchmarks: *sinring* (a) and *sintree* (b)

Table 2. Results obtained with PVM version 3.4.0

Bench.	Problem Size	PVM (sec)		
		Light	*Medium*	*Heavy*
ep	2^{28}	202	244	694
is	2^{23}-[0..2^{19}]	126	149	253
mg	256x256x256	645	1336	1976
sintree	20000 iterations	92	160	210
sinring	80000 iterations	203	255	377

where T_{pvm} (in seconds) is the execution time of one application in the original PVM environment and T_{sched} (in seconds) is the execution time of the same application in the DTS system.

- Local Overhead (LO): This metric was used to quantify the local workload overhead introduced by the execution of the parallel applications. One of the scripts used for the simulation of the Heavy workload(a compilation process) was taken as a reference. The LO metric is defined as follows:

$$LO = \frac{ET_{nondedicated} - ET_{dedicated}}{ET_{dedicated}} \quad (5)$$

where $ET_{dedicated}$ is the execution time spent by the compilation process in a dedicated workstation (86 s), and $ET_{nondedicated}$ is the execution time obtained by executing the script together with parallel applications.

4.1 Network Threshold

As the algorithm of Figure 4 shows, depending on the value of the network threshold, DTS activates the Dynamic or the Distributed mode of operation. In this section, the best threshold value has been studied. The results shown in the table 3 were obtained with our synthetic benchmark, *sinring*, whose network and CPU activity was under our control, since the sequential time of every process is an input argument of the benchmark. The experimentation was carried out with a medium owner workload.

The network activity of every node, showed in the column called *Net_Activity*, was obtained by measuring the average number of packets received and delivered by the pvm daemon. Table 3 shows the DTS gain, calculated in accordance with the formula 4, obtained with our benchmark with two different values of the

Table 3. DTS Gain according to the network threshold value

Bechmarks	Net_activity	Network Threshold					
		0	*300*	*600*	*900*	*1200*	*1500*
sinring	200	0.95	1.12	1.23	1.22	1.31	1.24
sinring	990	1.42	1.41	1.28	1.19	0.94	0.93

network activity when the threshold was varied between 0 and 1500 calls per second. As was expected, the results depend on the value of the threshold and the network activity of the benchmark. For a low communication benchmark case is better a high threshold whereas for a high communication benchmark is better a low threshold. Taking into account the above results, we have implemented DTS with a variable network threshold defined by the user.

4.2 DTS Performance

Table 4 shows the gain obtained in the execution of the three NAS benchmarks when they are executed in the DTS environment and with two different values of the IP period, 100 ms and 1000 ms. Table 5 shows the Local overhead in the execution of a compilation script when it is executed together with different parallel benchmarks. The behavior of the DTS environment can be determined comparing these two tables.

Table 4. Scheduler results

Bench.	Gain					
	IP=100ms			IP=1000ms		
	Light	*Medium*	*Heavy*	*Light*	*Medium*	*Heavy*
ep	0.91	1.13	1.26	0.96	1.14	1.24
is	0.99	1.09	1.12	1.01	1.15	1.21
mg	0.95	1.4	1.53	0.97	1.43	1.51

Table 5. Local Overhead obtained with DTS environment

Bench.	Local Overhead	
	IP=100 ms	*IP=1000 ms*
ep	2.25	2.65
is	0.75	0.80
mg	1.12	1.17

First of all, it is important to determine the IP value. An IP of 1000 ms increases the local overhead of interactive local tasks (see Table 5) even if there is a light distributed workload. On the other hand, a value of less than 100 ms

increases the overhead produced by the addition of context switches. Taking these two considerations into account, an IP value of 100 ms was chosen and it is used in the rest of the article.

From Table 4, it can be observed that in the *ep* case, a computing intensive benchmark, the results obtained in DTS with a light load are worse than in the PVM due to the additional overhead introduced by the DTS. On the other hand, since the DTS assigns to *ep* a better percentage of time than the PVM in the heavy case, the results are better.

Globally, for high message passing applications, *is* and *mg*, better results were obtained due to the synchronization between periods.

In general, the DTS environment gains with respect to the original PVM in the cases when there is almost some local load. Obviously, this gain is at the expense of the local overhead introduced by DTS environment as shown in Table 5, which, with the exception of the computing intensive distributed tasks, is very low and even it is reduced (*is* case).

4.3 Local Load Measurements

The performance of the distributed tasks can be improved at expense of a moderate local overhead, but how is the response time of the local applications affected?

With the aim of answering this question, comparison was made with the average response time of an implemented local benchmark which is executed jointly with one of the distributed benchmarks. The local benchmark continuously obtains the status of the standard output for printing by means of the *select* system call. The distributed benchmarks used were the *sintree* and *sinring* (message passing intensives) and *ep* (CPU bound). Table 6, shows the average and the maximum response time (max) in microseconds of three benchmarks, obtained in the two environments.

Table 6. Response time measurements (in μs). In the DTS case, IP = 100 ms

Bench.	PVM		DTS	
	average	*max*	*average*	*max*
sinring	6.13	201	5.93	247
sintree	6.01	454	5.94	477
ep	15.65	18043.7	29.81	20037.5

Table 6 shows that in any case the DTS system increases the response time of the local benchmark excessively. It is only slightly significant in the case of *ep*, but this response time overhead was not appreciated by the local user. Approximately the 98% of the collected samples were 5 or 6 μs (the response time of the *select* o.s. system call used in normal conditions), and only the 2% take higher values (due to the execution of the distributed benchmark), that

increases significantly the average. We can conclude that with an IP of 100 ms the overhead added for the distributed tasks does not damage the response time of local applications excessively.

4.4 Coscheduling Skew

The method used in implementing coscheduling is delivered by broadcasting a short message to all the nodes in the cluster. The PS and IS intervals must be synchronized between two pair of nodes but, due to the synchronization algorithm, some skew always exists between them. The *coscheduling skew* (δ) is the maximum out of phase between two arbitrary nodes, formally:

$$\delta = max(broadcast) - min(broadcast) \tag{6}$$

where *max(broadcast)* and *min(broadcast)* are the maximum and minimum time in sending a short broadcast message. We have mesured a coscheduling skew (δ) of 0.1 ms with the aid of the lmbench [13] benchmark. This value is insignificant in relation to the 100 ms of the IP interval. Thus, the *coscheduling skew* has no significant influence on the performance of the DTS system.

4.5 Context Switches

With the help of an implemented program (named *getcontext*) the context switch cost was measured in each workstation in function of the process size. The work done by *getcontext* was simulated as the summing up of a large size array before passing on one token (a short message) to the next process. The processes (a variable quantity) were connected in a ring of Unix pipes. The summing was an unrolled loop of about 2.7 thousand instructions. The effect was that both the data and the instruction cache was polluted to an extent before the token was passed on. Passing the size of a benchmark to *getcontext* as argument, it computed the context switch costs of that benchmark.

Table 7 shows the sizes of the measured benchmarks, *ep* and *sinring*, and their correspondent context switch costs. The benchmarks that fit in the main memory (the sum of its Resident Set Size and the memory of the o.s. applications $\leq$ 128 Mbytes) were chosen, and those that overlap it (*is* and *mg*) were discarded. This solution was adopted to avoid the extra overhead added by the swapping latency for large applications that may cause inaccuracies in our measurements.

Also, the number of context switches obtained with the *ep* and *sinring* benchmarks in the two environments (original PVM and DTS ($IP = 100ms$) with medium owner workload) were measured. The results are shown in Figure 6.

In Figure 6 it can be seen that in the case of the benchmark with high communications (*sinring*), the number of context switches is lower in the DTS case due to the synchronization between distributed tasks. In the case of *ep*, the number of context switches is increased by the DTS environment because *ep* must release CPU in each PS period. Thus, DTS system helps only the message passing applications.

Table 7. Sizes of the benchmarks: VI (Virtual Image: text + data + stack) and RSS (Resident Set Size) in Kbytes. Context switch costs (in μs) of one instance of the benchmark when one copy (1p.) or two (2 p.) were executed

Bench.	VI	RSS	1 p.	2 p.
ep	1160	596	2263	2902
sinring	1088	592	2182	2885

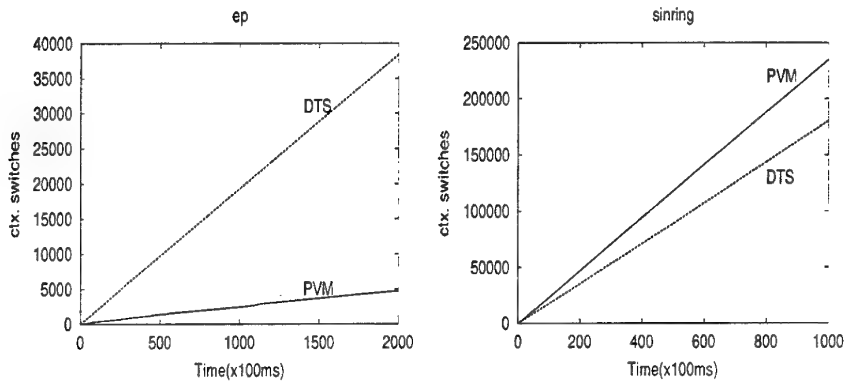


Fig. 6. *ep* and *sinring* context switches

5 Conclusions and Future Work

The DTS environment, which implements explicit coscheduling of distributed tasks in a non dedicated NOW has been introduced. Studying the communication architecture of the distributed applications and enabling each node of the system to change dynamically its configuration (dynamic and distributed), the communication performance of distributed applications was improved without damaging the performance of local ones excessively. Normally, the distributed tasks with high demand of CPU have not shown any improvement, but in these cases the overhead added to local tasks is not very significant.

We are interested in developing more efficient methods for synchronization, thus decreasing the overhead introduced in the coscheduling of distributed tasks.

In the future, we are interested in increasing the facilities of the DTS environment. The most important goal is to provide the DTS environment with the ability to run, manage and schedule more than one distributed applications and modify the scheduler algorithm according to this new functionality.

The dynamic mode algorithm of the DTS environment has a centralized nature. Furthermore the Console and the Master scheduler are localized in one module. If the Master module fails, the system goes down and there is no possibility of recovering the work done. Even if one node (not the Master) fails, the distributed application will stop abnormally and invalidate the execution of the

distributed application, perhaps for a long period of time. The solution is to provide the DTS environment fully distributed behavior. For the accomplishment of these objectives we have to study and propose new algorithms to implement fault tolerance.

References

1. Anderson, T., Culler, D., Patterson, D. and the Now team: A case for NOW (Networks of Workstations). IEEE Micro (1995) 54–64
2. Litzkow, M., Livny, M., Mutka, M.: Condor - A Hunter of Idle Workstations. Proceedings of the 8th Int'l Conference of Distributed Computing Systems (1988) 104–111
3. Russ, S., Robinson, J., Flachs, B., Heckel, B.: The Hector Distributed Run-Time Environment. IEEE trans. on Parallel and Distributed Systems, Vol.9 (11), (1988)
4. Du, X., Zhang. X.: Coordinating Parallel Processes on Networks of Workstations. Journal of Parallel and Distributed Computing (1997)
5. Crovella, M. et al.: Multiprogramming on Multiprocessors. Proceedings of 3rd IEEE Symposium on Parallel and Distributed Processing (1994) 590–597
6. Dusseau, A., Arpaci, R., Culler, D.: Effective Distributed Scheduling of Parallel Workloads. ACM SIGMETRICS'96 (1996)
7. Arpaci. R., Dusseau, A., Vahdat, A., Liu, L., Anderson, T., Patterson, D.: The Interaction of Parallel and Sequential Workloads on a Network of Workstations. ACM SIGMETRICS'95 (1995)
8. Zhang. X., Yan Y.: Modeling and Characterizing Parallel Computing Performance on Heterogeneous Networks of Workstations. Proc. Seventh IEEE Symp. Parallel and Distributed Processing (1995) 25–34
9. Ousterhout, J.: Scheduling Techniques for Concurrent Systems. Third International Conference on Distributed Computing Systems (1982) 22–30
10. Solsona. F., Gine, F., Hernández, P., Luque, E.: Synchronization Methods in Distributed Processing. Proceedings of the Seventeenth IASTED International Conference. Applied Informatics (1999) 471–473
11. Ferrari, D., Zhou, S.: An Empirical Investigation of Load Indices for Load Balancing Applications. Proc. Performance '87, 12th Int'l Symp. Computer Performance Modeling, Measurement, and Evaluation. North-Holland. Amsterdam (1987) 515–528
12. Parkbench Committe: Parkbench 2.0. http://www.netlib.org/park-bench (1996)
13. Mc. Voy, L., Staelin, C.: lmbench: Portable tools for performance analysis. Silicon Graphics. Inc, ftp://ftp.sgi.com/pub/lm-bench.tgz (1997)

An Approximation Algorithm for the Static Task Scheduling on Multiprocessors

Janez Brest, Jaka Jejčič, Aleksander Vreže, and Viljem Žumer

University of Maribor
Faculty of Electrical Engineering and Computer Science
Smetanova 17, 2000 Maribor, Slovenia
janez.brest@uni-mb.si

Abstract. In this paper we proposed a new multiprocessor scheduling algorithm, called MCP/AM, which is based on well known MCP algorithm. Both, the MCP/AM and MCP algorithms have the same complexity of $O(v^2 \log v)$, where v is the number of nodes in the task graph. Algorithm, described in this paper, does not model communication time, that is, it assumed that the data transmissions between processors did not take any time.
The MCP/AM algorithm outperforms two other algorithms (ETF and MCP) by generating similar or better solution in the term of the scheduling length.

Keywords: Parallel processing, directed acyclic graph, task scheduling, compiler.

1 Introduction

Multiprocessor systems [9, 16, 6, 17] are increasingly being used to meet the high performance and intense computation needs of today's applications.

To efficiently execute a program on a multiprocessor system, it is essential to solve a minimum execution time multiprocessor scheduling problem [13, 11], which determines how to assign a set of tasks to processors and in what order those tasks should be executed to obtain the minimum execution time.

The tasks can then be scheduled to the processors for execution by using a suitable scheduling algorithm, static in compile-time or dynamic in run-time [5, 7, 3]. In this paper static scheduling is discussed.

Static scheduling, except for a few highly simplified cases, is an NP-complete problem. Thus, heuristic approaches are generally sought to tackle the problem. Traditional static scheduling algorithms attempt to minimize the schedule length through iterative local minimization of the start times of individual tasks.

On the other hand for example the Dynamic Level Scheduling (DLS) algorithm dynamically selects tasks during the scheduling process [12]. However, like most greedy algorithms, these scheduling approaches cannot avoid making a local decision which may lead to an unnecessarily long final schedule.

Although static scheduling is done at compile-time and therefore can afford some extra time in generating a better solution, back-tracking techniques are not employed to avoid high complexity.

The scheduling problem is intractable even when severe restrictions are imposed on the task graph and the machine model. As optimal scheduling of tasks is a strong NP-hard problem, many heuristic algorithms have been introduced in the literature [4]. The following simplifying assumptions about the task graph and the machine model are common in *Bounded Number of Processor (BNP) Scheduling* [2]:

1. the communication costs on the edges are zero;
2. the processors are fully connected;
3. the processors are homogeneous, that is, their processing speeds are the same.

Due to the intractability of the problem, heuristics are devised for obtaining suboptimal solutions in an affordable amount of computation time. Even though most heuristics can produce high quality solutions, their time complexities are quite high. Furthermore, heuristics designed with more relaxed assumptions tend to incur higher time complexities. Thus, many heuristics work well for small task graphs but do not scale well with the problem size. Therefore, the solution quality and applicability are usually in conflict with the goal of reducing the time complexity [2].

In this paper we proposed a low time complexity multiprocessor scheduling algorithm, called MCP/AM, which is based on critical path (CP) algorithm, such as, for example, the MCP [18] algorithm. It generates high quality scheduling solutions.

The remaining paper is organized as follows: In the next section, we present a brief overview of various approaches that have been proposed for the DAG scheduling problem. In Sect. 3, we present the proposed algorithm, and discuss its design principles. Section 4 includes some scheduling examples illustrating the operation of the algorithm. We present the experimental results in Sect. 5, and conclude the paper with some final remarks in Sect. 6.

2 The Multiprocessor Scheduling Problem

In static scheduling, we represent a parallel program by a directed acyclic graph (DAG) [6, 16, 9, 15]. In a DAG, $G = (V, E)$, V is a set of v nodes, representing the tasks, and E is a set of e directed edges, representing the communication messages. Edges in a DAG are directed and, thus, capture the precedence constraints among the tasks. The cost of node n_i, denoted as $w(n_i)$, represents the computation cost of the task. The cost of the edge, emerges from the source node n_i and incidents on the destination node n_j, denoted by c_{ij}, represents the communication cost of the message.

The source node of an edge is called a parent node, while the destination node is called a child node. A node with no parent is called an entry node and a

node with no child is called an exit node. A node can only start execution after it has gathered all of the messages from its parent nodes. The *b-level* of a node is the length (sum of the computation costs only) of the longest path from this node to an exit node. The *t-level* of a node is the length of the longest path from an entry node to this node (excluding the cost of this node).

Figure 3 shows an example of the DAG which we will use in the subsequent discussion.

The objective of scheduling is to minimize the schedule length, which is defined as the maximum finish time of all the nodes, by properly assigning tasks to processors such that the precedence constraints are preserved.

The existing scheduling algorithms are classified into four categories by Ahmad and Kwok [2]:

1. Bounded Number of Processors (BNP) Scheduling: A BNP algorithm schedules a DAG to a limited number of processors directly. The processors are assumed to be fully connected without any regard to link contention and scheduling of messages. The proposed algorithm belongs to this class.
2. Unbounded Number of Clusters (UNC) Scheduling: A UNC algorithm schedules a DAG to an unbounded number of clusters. The clusters generated by these algorithms may be mapped onto the processors using a separate mapping algorithm. These algorithms assume the processors to be fully connected.
3. Arbitrary Processor Network (APN) Scheduling: An APN algorithm performs scheduling and mapping on an architecture in which the processors are connected via a network topology. An APN algorithm also explicitly schedules communication messages on the network channels, taking care of the link contention factor.
4. Task-Duplication-Based (TDB) Scheduling: A TDB algorithm duplicates tasks in order to reduce the communication overhead. Duplication, however, can be used in any of the other three classes of algorithms.

For our purpose, we will compare the proposed algorithm with two other BNP algorithms (ETF and MCP).

The proposed algorithm is based on the classic *list scheduling* technique [13, 1]. The basic idea of list scheduling is to make a scheduling list (a sequence of nodes for scheduling) by assigning them some priorities, and then repeatedly execute the following two steps until all the nodes in the graph are scheduled: (1) Remove the first node from the scheduling list; (2) Allocate the node to a processor which allows the earliest start time.

In a traditional scheduling algorithm, the scheduling list is statically constructed before node allocation begins, and, more importantly, the sequencing in the list is not modified.

The ETF Algorithm

The Earliest Task First (ETF) algorithm [8] uses static node priorities and assumes only a bonded number of processors [13, 14]. At each scheduling step, the

ETF algorithm first computes the earliest start times for all the *ready* nodes and then selects the one with the smallest value of the earliest start time. A node is ready if all its parent nodes have been scheduled. The earliest start time of a node is computed by examining the start time of the node on all processors exhaustively. When two nodes have the same value of the earliest start times, the ETF algorithm breaks the tie by scheduling the one with a higher static priority. The complexity of the ETF algorithm is $O(pv^2)$, where p is the number of the processing elements in the target machine, and v in the number of nodes in the task graph.

The MCP Algorithm

Similar to the ETF algorithm, the Modified Critical Path (MCP) algorithm [18] constructs a list of tasks before the scheduling process starts.

The MCP algorithm uses the ALAP (As-Late-As-Possible) start time of a node as the scheduling priority. The MCP algorithm first computes the ALAPs of all the nodes, then constructs a list of nodes in ascending order of ALAP times. Ties are broken by considering the ALAP times of the children of a node. The MCP algorithm then schedules the nodes on the list one by one so that a node is scheduled to a processor that allows the earliest start time using the insertion approach. The MCP algorithm looks for an idle time slot for a given node. The algorithm is briefly described in Fig. 1 [18, 13].

(1) Compute the ALAP time of each node.
(2) For each node, create a list which consists of the ALAP times of the node itself and all its children in descending order.
(3) Sort these lists in ascending lexicographical order. Create a node list according to this order.

Repeat
(4) Schedule the first node in the node list to a processor that allows the earliest execution, using the insertion approach.
(5) Remove the node from the node list.
Until the node list is empty.

Fig. 1. The MCP algorithm.

The complexity of the MCP algorithm is $O(v^2 \log v)$.

3 The Approximation Algorithm

In this section we present the proposed scheduling MCP/AM algorithm.

The multiprocessor scheduling problem treated in this paper is to determine a non-preemptive schedule to minimize the execution time (or the schedule length)

when a set of v computational tasks having arbitrary precedence constraints and arbitrary processing time are assigned to p processors of the same ability for execution. These tasks are represented by a task graph (DAG) as shown in Fig. 3.

It is assumed that the target platform is a multiprocessor system and the communication costs between nodes are zeros.

Table 1 summarizes the definitions of the notations used throughout the paper.

Table 1. Definitions of Notations

Notation	Definition
n_i	A node in the parallel program task graph
$w(n_i)$	The computation cost of node n_i
c_{ij}	The communication cost of the directed edge from node n_i to n_j
v	The total number of nodes in the task graph
e	The total number of edges in the task graph
p	The number of processing elements in the target multiprocessor system
CP	A critical path of the task graph
EST	The earliest start time
ALAP	The As-Late-As-Possible start time
$SL_j(i)$	The schedule length of the step i of the scheduling process on the processor j

A node can be scheduled to a processor if the processor has an idle time slot that starts later than the node's parents finish times and is large enough to accommodate the node. The simple procedure in [13] outlines the computation of the start time of a node on a processor.

In the following, we discuss some of the principles used in the design of our algorithm. To minimize the final schedule length, we select a node as it is selected in the MCP algorithm, which is described in Sect. 2. At each step of the scheduling process, the first node is removed from the list of nodes (list of nodes is sorted in increasing lexicographical order of the latest possible start times) and it is scheduled to a processor.

While we are able to identify a selected node, we still need a method to select an appropriate processor for scheduling that node into the most suitable idle time slot. At each step, the algorithm needs to find the most suitable processor which contains the most suitable place in time for a selected node.

The MCP algorithm schedules the selected node to a processor that allows for the earliest start time. Our MCP/AM algorithm has another processor selection criteria and it is described in Fig. 2.

The function *Build_ALAP*() computes the ALAP time of each node and create a list, which consists of the ALAP times of the node itself and all its chil-

```
Build_ALAP():                          // As late as possible time for each node
Sort_ALAP():                           // See the MCP algorithm
for (i = 0; i < v; i++)                // v is number of tasks
{
  t_i = EST(ALAP(n_i));                // Earliest start time of node n_i in a
                                       ALAP list
                                       // SL_j(i) is schedule length of the step i
                                       of the scheduling process on the processor
                                       j
  if a processor j exists where        // Processor selection
    SL_j(i) = t_i
  then
    schedule node n_i to the processor j
  else
    schedule node n_i to a processor that
    allows the earliest execution
}
```

Fig. 2. The MCP/AM algorithm.

dren in descending order. Function $Sort\_ALAP()$ sorts these lists in ascending lexicographical order as in the MCP algorithm.

Assumed that. in the scheduling process there are already scheduled $i-1$ nodes. Next selected node is n_i. Our MCP/AM algorithm tries to find a processor j for the selected node n_i. We need to distinguish two cases of the processor selection step. If a processor exists, say j, which satisfy that $SL_j(i)$ is equal to the earliest start time of the selected node n_i, our algorithm assigns the selected node n_i to the processor j. Otherwise it assigns the selected node n_i to a processor that allows the earliest execution (like the MCP algorithm).

The complexity of the MCP/AM algorithm is $O(v^2 \log v)$ – the MCP algorithm has the equal complexity.

4 Scheduling Example

In this section, we present an example to demonstrate the operation of the proposed algorithm using the task graph shown in Fig. 3. The task graph was drawn with the Graphlet Tool[1].

The schedules of the ETF, MCP, and MCP/AM algorithms are shown in Fig. 4. The entry and exit node are dummy. The MCP algorithm, as mentioned above, creates a list of edges and schedules the task graph onto the multiprocessor machine with 2 processors (processing elements) in the order: $n_1, n_2, n_5, n_3, n_8, n_7, n_4, n_{11}, n_{10}, n_9, n_6, n_{12}$. The MCP/AM schedules the nodes in the same order as the MCP algorithm. The ETF algorithm schedules the nodes in the order: $n_1, n_2, n_5, n_3, n_4, n_7, n_8, n_{10}, n_6, n_9, n_{11}, n_{12}$.

[1] *http://www.fmi.uni-passau.de/Graphlet*

The schedule lengths generated by the ETF, MCP, and MCP/AM algorithm, are 67, 64, and 63 time units, respectively.

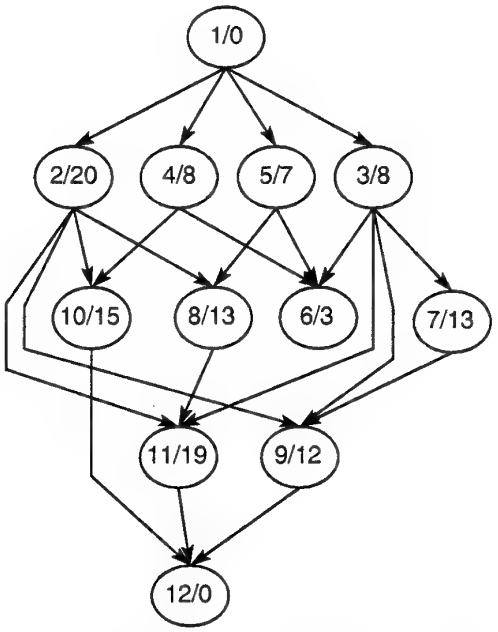


Fig. 3. An example of a task graph with 12 nodes

5 Results

In this section, we present the performance results of the MCP/AM algorithm and compare it with the ETF and MCP algorithms.

We have implemented the scheduling algorithms on a SUN workstation. They were evaluated by using a Prototype Standard Task Graph Set[2]. The Prototype Standard Task Graph Set has 300 task graphs with 50 to 2500 tasks.

The results obtained in our experiments are shown in Table 2. The second column indicates the name of the task graph instance. In next three columns results of the scheduling length for the ETF, MCP, and MCP/AM algorithm, respectively, are shown. The best schedule length value of all the algorithms is boldface. In the last two columns the optimal scheduling length value and difference between optimal scheduling length and the best schedule length value

[2] *http://www.kasahara.elec.waseda.ac.jp/schedule/*

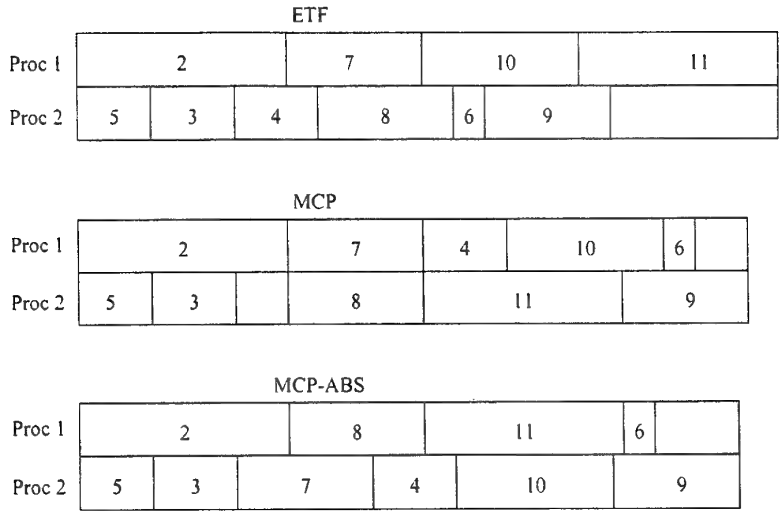


Fig. 4. The schedules of the task graph on Fig. 3 generated by the ETF (schedule length = 67 time units), MCP (schedule length = 64 time units) and MCP/AM algorithms (schedule length = 63 time units).

of all the algorithms are shown, respectively. For some problem instances, the optimal scheduling length is not known.

In order to rank all the algorithms in terms of the scheduling lengths, we made a global comparison. We observed the number of times each algorithm performed better, worse or the same compared to each of the other two algorithms. This comparison is presented in Fig. 5. Here, three boxes have the left and the right side. Each left side of the box compares two algorithms – the algorithm on the left side and the algorithm on the top. Each left side of the box contains three numbers preceded by ">", "<", and "=" signs which indicate the number of times the algorithm on the left performed better, worse, or the same, respectively, compared to the algorithm shown on the top. Each comparison is based on the total of 300 task graphs.

Each right side of the box contains the number of times when one of algorithms, the algorithm on the left side and the algorithm on the top, find the optimal scheduling length. Optimal scheduling lengths are known for 255 of all 300 task graphs. They were computed on a parallel machine with 3000 processors using ISH algorithm [11, 10].

For example, the MCP/AM algorithm performed better than the MCP algorithm in 246 cases, performed worse in 54 cases, and never performed the same. The MCP/AM algorithm or the MCP algorithm or both of them found optimal solution of the scheduling length in 156 cases. Similarly, the MCP algorithm performed better than the ETF algorithm in 205 cases, performed worse in 51

Table 2. Schedule results of 40 task graph instances

	Graph	ETF	MCP	MCP/AM	Optimum	Error
1	proto000.stg	**537**	**537**	**537**	537	0
2	proto001.stg	1191	1179	**1178**	1178	0
3	proto002.stg	357	363	**347**	341	6
4	proto003.stg	**556**	**556**	**556**	556	0
5	proto004.stg	267	238	**222**	—	—
6	proto005.stg	758	749	**742**	742	0
7	proto006.stg	171	154	**143**	—	—
8	proto007.stg	492	**489**	**489**	489	0
9	proto008.stg	578	582	**572**	571	1
10	proto009.stg	**625**	**625**	**625**	625	0
11	proto010.stg	351	338	**334**	334	0
12	proto011.stg	513	496	**485**	—	—
13	proto012.stg	1804	1795	**1793**	1793	0
14	proto013.stg	698	688	**682**	681	1
15	proto014.stg	523	520	**511**	—	—
16	proto015.stg	513	513	**494**	491	3
17	proto016.stg	1016	1026	**1007**	—	—
18	proto017.stg	487	475	**463**	—	—
19	proto018.stg	704	706	**701**	700	1
20	proto019.stg	683	682	**668**	667	1
21	proto020.stg	1523	1514	**1505**	1504	1
22	proto021.stg	644	632	**608**	605	3
23	proto022.stg	1625	1620	**1612**	1609	3
24	proto023.stg	1619	1628	**1614**	1612	2
25	proto024.stg	1295	1291	**1283**	1281	2
26	proto025.stg	1193	1198	**1192**	1188	4
27	proto026.stg	1509	1502	**1500**	1500	0
28	proto027.stg	2003	**2001**	**2001**	2000	1
29	proto028.stg	1538	1506	**1504**	1504	0
30	proto029.stg	845	**830**	**830**	830	0
31	proto030.stg	1076	1057	**1051**	1051	0
32	proto031.stg	1383	1364	**1356**	—	—
33	proto032.stg	1411	1396	**1389**	—	—
34	proto033.stg	3690	3688	**3685**	3685	0
35	proto034.stg	1515	1517	**1507**	1507	0
36	proto035.stg	3852	3845	**3842**	3841	1
37	proto036.stg	987	974	**966**	966	0
38	proto037.stg	4090	4089	**4087**	4086	1
39	proto038.stg	1631	1621	**1617**	1616	1
40	proto039.stg	3167	3162	**3159**	3159	0

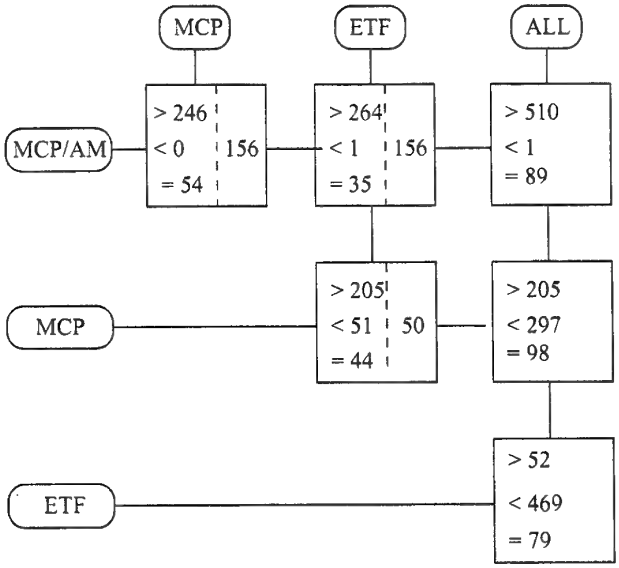


Fig. 5. A global comparison of the three algorithms in terms of better, worse, and equal performance.

cases, and performed the same in 44 cases. Algorithms combined found optimal scheduling length in 50 cases.

An additional box for each algorithm compares that algorithm with all other algorithms combined.

The experimental results of the quality of the schedule length are summarized in Table 3. The MCP/AM algorithm found the optimal schedule length in 156 cases, and the solution within 5% in 98 cases.

We can notice that the proposed MCP/AM algorithm outperformed two other well known algorithms. Based on these experiments, we can order all three algorithms in the following order: MCP/AM, MCP, and ETF. The same order of the MCP and ETF algorithms can be found in [14], where communications are assumed among the tasks.

6 Conclusion and Future Work

This paper presents a task scheduling algorithm which can schedule directed acyclic graphs (DAGs) with a complexity of $O(v^2 \log v)$, where v is the number of tasks in the DAG.

The performance of the algorithm has been observed by comparing it with other existing bounded number of processor (BNP) scheduling algorithms in terms of the schedule length.

Table 3. Schedule lengths with respect to optimal solutions

	Quality of the solution (Error)	ETF	MCP	MCP/AM
1	0% (optimum)	33	50	156
2	< 5%	178	182	98
3	5% - 10%	30	21	1
4	10% - 15%	11	2	0
5	15% - 20%	3	0	0
6	> 20%	0	0	0
7	Optimum not known	45	45	45
	Total	300	300	300

The algorithm schedules the tasks and is suitable for graphs with arbitrary computation and without communication costs, and is applicable to systems with homogeneous fully connected processors.

In the future we intent to extend our algorithm to schedule both the tasks and messages for task graphs with arbitrary computation and communication costs, and it will be applicable to systems with arbitrary network topologies using homogeneous or heterogeneous processors.

Acknowledgments

This research was supported by the Ministry of Science and Technology, Republic of Slovenia under contact number J2-7502-796.

References

1. Thomas L. Adam, K. M. Chandy, and J. R. Dickson. A comparison of list schedules for parallel processing systems. *Communications of the ACM*, 17(12):685–690, December 1974.
2. I. Ahmad and Y.-K. Kwok. On parallelizing the multiprocessor scheduling problem. *IEEETPDS: IEEE Transactions on Parallel and Distributed Systems*, 10, 1999.
3. Janez Brest, Viljem Žumer, and Milan Ojsteršek. Dynamic scheduling on a network heterogeneous computer system. In *4th International ACPC Conference Including Special Tracks on Parallel Numerics (ParNum'99) and Parallel Computing in Image Processing, Video Processing, and Multimedia; LNCS 1557*, pages 584–585, Salzburg, Austria, 1999.
4. D. Darbha and D. P. Agrawal. Optimal scheduling algorithm for distributed-memory machines. *IEEETPDS: IEEE Transactions on Parallel and Distributed Systems*, 9, 1998.
5. Mary. M. Eshagian, editor. *Heterogeneous Computing.* Artech House, Inc., Norwood, MA 02062, ISBN 0-89006-552-7, 1996.
6. Ian Foster. *Designing and Building Parallel Programs.* Addison-Wesley, ISBN 0-201-57594-9, 1995.

7. Emile Haddan. Load Balancing and Scheduling in Network Heterogeneous Computing. In Mary. M. Eshagian, editor, *Heterogeneous Computing*, pages 224–276. Norwood, MA 02062, ISBN 0-89006-552-7, 1996. Artech House, Inc.
8. Jing Jang Hwang, Yuan-Chieh Chow, Frank D. Anger, and Chung-Yee Lee. Scheduling precedence graphs in systems with interprocessor communication times. *SIAM Journal on Computing*, 18(2):244–257, April 1989.
9. K. Hwang and Z. Xu. *Advanced Computer Architecture: Technology, Architecture, Programming.* McGraw-Hill, New York, 1998.
10. H. Kasahara. H. Honda, and S. Narita. Parallel processing of near fine grain tasks using static scheduling on OSCAR (optimally scheduled advanced multiprocessor). In IEEE, editor, *Proceedings, Supercomputing '90: November 12–16, 1990, New York Hilton at Rockefeller Center, New York, New York*, pages 856–864, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1990. IEEE Computer Society Press.
11. H. Kasahara and S. Narita. Practical multiprocessor scheduling algorithms for efficient parallel processing. *IEEE Trans. on Computers*, 33(11):1023, November 1984.
12. Y.-K. Kwok and I. Ahmad. FASTEST: A practical low-complexity algorithm for compile-time assignment of parallel programs to multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 10(2):147–159, February 1999.
13. Y.-K. Kwok and I. Ahmad. Parallel program scheduling technique. In Buyya Raykumar, editor, *High Performance Cluster Computing: Architectures and Systems.* Prentice Hall - PTR, NJ, USA, 1999.
14. Yu-Kwong Kwok and Ishfaq Ahmad. Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors:. *IEEE Transactions on Parallel and Distributed Systems*, 7(5):506–521, May 1996.
15. M. Quinn. *Parallel Computing: Theory and Practice.* McGraw-Hill, 1994.
16. Buyya Raykumar, editor. *High Performance Cluster Computing: Architectures and Systems.* Prentice Hall - PTR, NJ, USA, 1999.
17. Barry Wilkinson and Michael Allen. *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers.* Prentice-Hall. Englewood Cliffs, NJ 07632, USA, 1998.
18. Min-You Wu and Daniel D. Gajski. Hypertool: A programming aid for message-passing systems. *IEEE Transactions on Parallel and Distributed Systems*, 1(3):330–343, July 1990.

A New Buffer Management Scheme for Sequential and Looping Reference Pattern Applications†

Jun-Young Cho, Gyeong-Hun Kim, Hong-Kyu Kang and Myong-Soon Park

Dept. of Computer Science & Engineering, Korea University,
Sungbuk-ku, Seoul 136-701, Korea
{jycho, kgh, hkkang, myongsp}@iLab.korea.ac.kr

Abstract. MRU replacement policy is frequently used to improve the performance of buffer caching for sequential and looping pattern applications. On the way of implementing MRU on Linux, we observed that MRU shows lower response time by up to 100% compared to LRU. Indirect blocks, which are used in the file structure of Unix family operating systems for large-size file, are the main reason of decreasing performance. Indirect blocks are fetched but immediately replaced by MRU replacement policy, even those will be soon and frequently needed again. Based on this observation, we propose a buffer replacement policy named 'LMRU'. LMRU maintains frequently-used blocks such as indirect blocks in the cache, even it manages all other blocks on buffer cache with MRU. We have designed and implemented it in a Linux kernel. LMRU improves the response time by up to 70% compared to LRU and 163% compared to MRU.

1. Introduction

Rapid improvements in processor and memory performance have created a situation in which the file system I/O has become a major bottleneck for system performance[10]. To solve this problem, a lot of researches have been done in the areas of virtual memory, file system and I/O system. The management of the buffer cache is one of these research areas.

Since the memory allocated to buffer cache is limited, a block should be replaced to store a new data block. A buffer replacement policy is the problem of deciding which memory block to replace when making room for a new one. An optimal buffer replacement algorithm is one that incurs the lowest number of cache misses. However, this algorithm requires future knowledge of the reference sequence, and it is not realizable in general. As a result, storage systems employ a number of buffer replacement algorithms, which attempt to approximate the performance behavior of the optimal buffer replacement algorithm[11]. LRU algorithm is one of the most popular replacement policies, particularly in commercial implementations. It is used

† This research was supported by KOSEF under grant No. 96-0101-04-01-3

widely in database buffer management, virtual-memory management, file system and I/O caches. This policy exploits the principle of temporal locality and evicts the block used least in the recent past on the assumption that it will not be used in the near future. However it is well known that an application which shows large sequential or looping reference pattern is not fit to use LRU. In prior works, MRU is used to manage such patterns. MRU is the most appropriate policy to manage such patterns[1][2][3][4].

To effectively support a large sequential or looping reference patterns of some applications, we implemented MRU in Linux buffer cache. When we were testing our implementation, we observed that the same block was fetched and replaced repeatedly. It causes lower performance even though the application's reference pattern is suitable to MRU. We found that indirect blocks, which are used in file system of Unix family operating systems for large size file, are the main reason of decreasing performance. When an application access a data block, larger than 12 block, it must refer the block via indirect block, as a result indirect block is fetched and replaced frequently when we applied MRU policy.

LMRU solves this problem. It maintains frequently-used blocks such as indirect blocks in the buffer cache, even it manages all other blocks on the buffer cache with MRU. LMRU improves the response time by up to 70% compared to LRU and 163% compared to MRU.

In Section 2, we discuss background. In Section 3, we describe the indirect block in the Unix file system. In Section 4, we propose a new buffer cache management policy for sequential or looping reference pattern application. In Section 5, we analysis an implementation and the performance of the LMRU. Finally, we present our conclusions in Section 6.

2. Background

In this Section we introduce the buffer cache management scheme, present an operation and analysis of MRU and look into data access pattern.

2.1 Buffer Cache

The block-device interface uses the buffer cache to minimize the number of I/O requests that require an I/O operation, and to synchronize with file system operations on the same device[8] . When a process tries to read data from the disk, the OS kernel gives the request to the buffer cache module. The buffer cache searches its buffer pool to see if the requested data is in the pool. If valid data is found, the data is returned to process without accessing the disks. If there is no buffer related to the requested data, new free buffer is allocated to store for the data from free list.

If there is no free buffer to use, some of buffers should be replaced. When buffer cache module select buffers to be flushed, it is well known that LRU policy works well for general cases. Traditional OS uses LRU replacement policy.

2.2 MRU (Most Recently Used)

MRU keeps track of the last time each block was accessed and replaces the block that has been accessed in the nearest time. Among all buffer replacement policies, it has been known that for looping reference, a MRU replacement requires the fewest number of faults[4]. So in previous works, MRU is used to manage large sequential or looping reference pattern [1][2][3][4]. [3] proposed SEQ. SEQ normally performs LRU replacement. When SEQ detects sequential access pattern, SEQ performs MRU replacement on the sequence to prevent LRU list flooding. [1][2][4]applies MRU as a buffer replacement policy to manage long sequential or looping reference patterns. In this paper we analyze what parameter has influence on the performance of the system when we apply MRU as a buffer management policy. Let the size of buffer cache be S, loop length L, the number of loop iteration K(where S>L, K>0).

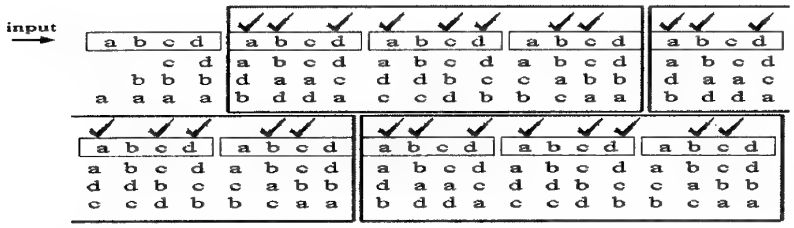


Fig. 1. An Example of mini-cycle when we manage looping data as MRU.

As figure 1 shows, when we manage MRU for sequential and looping reference patterns, it occurs in a periodic pattern. The length of a period is L*(L-1). We will call the period mini-cycle. When we observe the number of hits in a mini-cycle, there are S or S-1 hits occurring. So the number of hits in the mini-cycle are as follows.

The number of hits in a mini-cycle

$$=S\times(S-1)+(S-1)\times(L-S)=(S-1)\times L \tag{1}$$

Let the number of mini-cycles be M. Then M can be represented as $M = \lfloor (K-1)/(L-1) \rfloor$. Let R is the value of (K-1) modulo (L-1) then the total number of hit is as follows.

$$\text{Number of hits} = \begin{cases} \{(S-1)\times L\}\times M + S\times R & \text{if } R \le (S-1) \\ \{(S-1)\times L\}\times M + (S-1)\times(R+1) & \text{if } R > (S-1) \end{cases} \tag{2}$$

Equation (2) shows us that the performance of the system has related with the size of buffer cache, loop length and the number of iteration when we apply MRU as a buffer cache management scheme.

2.3 Data Access Pattern

Recent research has shown that most applications show various block reference patterns. Applications for continuous media generally show a sequential or periodic

reference pattern[5]. A large class of scientific applications show a looping reference pattern[6]. Database applications show a irregular block reference pattern[7]. Based on the previous research, we classify the block reference pattern as follows.

- **Sequential reference**: A sequential reference pattern has the property that all data blocks are referenced one after another.
- **Looping reference**: A looping reference pattern has the property that sequential reference is performed repeatedly.
- **Irregular reference**: An irregular reference pattern is not equal to reference probability among all blocks.

3. Indirect Block in the Unix File System

In Unix, the information required for management is kept strictly apart from the data collected in a separate inode structure for each file. Inode contains information about the file size, its location, owner of the file, time of creation, and so on. In addition to descriptive information about the file, the inode contains the pointer to a block of pointers to additional data blocks. BSD and Linux keep 15 pointers of the inode block in the file's inode block[8]. The first 12 of these pointers point to direct blocks. Thus, the data for small (no more than 12 blocks) files do not need a separate index block. The next three pointers point to indirect blocks. The first indirect block is an index block, containing not data but rather the pointers of blocks that contain data. Then there is a double indirect block pointer, which contains the pointer of a block that contains the pointers of blocks that contain pointers to the actual data blocks. The last pointer contains the pointer of a triple indirect block. Since an application accesses a data block it must access data block via indirect block, an indirect block is fetched and replaced if we use buffer cache management policy as MRU.

Figure 2 shows a referenced block in the buffer cache when an application reads a file sequentially in Linux. It shows that data block is referenced via indirect block.

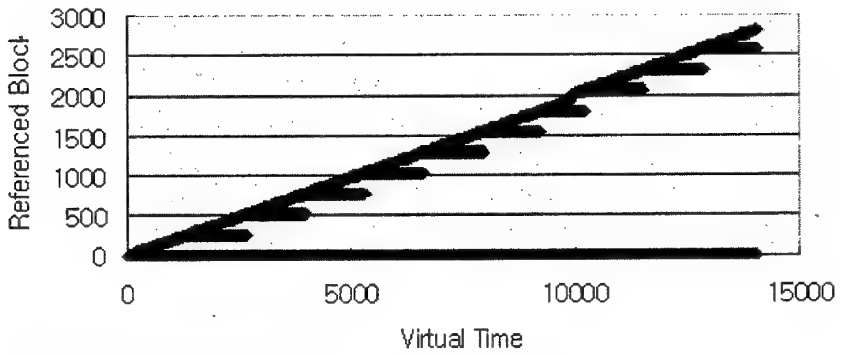


Fig. 2. Block reference pattern as we read a file sequentially

In this paper, we analyze how many indirect blocks are referenced as we read a file sequentially in order to know what parameters are related to indirect blocks. Let file size be FS, block size BS, the size of pointer K. Then we can represent the number of blocks N in a file as $N = \lceil FS/BS \rceil$ and the number of pointers P in a block as $P = \lceil BS/K \rceil$. We assume that an inode can access data block directly up to D^{th} block and an inode can support triple indirect block. Then the number of referenced indirect blocks is as follows.

The number of referenced indirect blocks

$$= \begin{cases} 0 & 0 \le N \le D \\ N - D & D < N \le (D + P) \\ 2N - P - 2D & (D + P) < N \le (D + P^2 + P) \\ 3N - 2P^2 - 2P - 3D & (D + P^2 + P) < N \le (D + P^3 + P^2 + P) \end{cases} \quad (3)$$

Equation (3) shows us that the number of indirect blocks relates to the number of blocks in a file. As a file size gets larger, indirect blocks are referenced more and more.

4. LMRU Algorithm

When we apply MRU in large sequential or looping reference pattern, the indirect block is fetched and replaced frequently. To solve this problem we propose LMRU. LMRU is designed to consider characteristics of indirect block and data block in large sequential and looping reference patterns. To manage such patterns, we have two regions IR(region for filtering indirect block) and DR(region for managing data block). Each region is composed of list structure. A block which is located at the head of the list has the highest priority to replace and tail of the list has the lowest priority to replace. Every block is listed in hash table so it is possible to O(1) time search. The LMRU has the following data structure.

- **IR**: When a block hits in IR, then the block moves to the tail of the list. When there is no free buffer for assigning to a new block, LMRU replaces a block at the head of the list and then puts the block at the tail of the list.
- **DR**: When a block hits in DR then the block moves to the head of the list. When there is no free buffer for assigning to a new block, LMRU replaces a block at the head of the list and then puts the block at the head of the list.

LMRU has a two-tiered structure. When a block is fetched from a disk, LMRU puts it into IR.When a block is no longer needed in IR, then the block moves into DR. Figure 3 depicts the logical flow of LMRU.

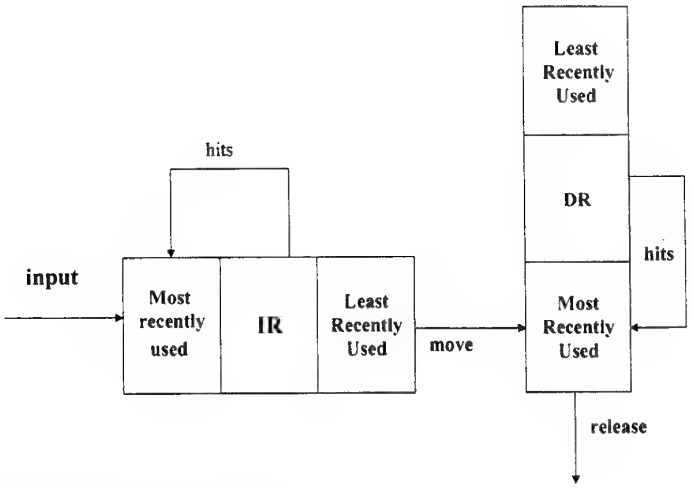


Fig. 3. The logical flow of LMRU

IR is designed to maintain frequently-used blocks such as indirect blocks and DR is designed to manage data block. So LMRU is a structure to cache indirect block and manage data as MRU.

5. Performance Measurements

We present the results of experiments. We used Linux-2.0.32 on a 200MHZ Intel Pentium PC with 64MB RAM and 5GB Quantum Fireball hard disk. The size of Linux buffer cache grows dynamically. We fix the size of buffer cache in order to test the effect of various replacement policies as we increase the size of buffer cache.

We experimented to decide the size of IR and then compare the performance of various applications under LMRU, LRU and MRU. The application traces are used in [1][2].

5.1 File Access Traces

Applications we used are :

- **Dinero**: Dinero is a cache simulator. Dinero reads a trace file sequentially and repeatedly.
- **Cscope**: Cscope is a C-source examination tool. It builds a database of all source files, then uses the database to answer queries about the program.
- **Glimpse**: Glimpse is a text information retrieval system. It uses two-level searching. First it searches the index for a list of all blocks sequentially that may contain a match to the query. Then, it searches each such block separately.

- **Sort**: Sort is an external sorting utility in Unix. We used a 17MB text file as input, and sorted numerically on the first field.

The applications that we used are summarized in Table1.

Program	Description	The size of input data	Access pattern
Dinero	Cache simulator	8M	Sequential, Looping
Cscope	C examination tool	18M	Sequential, Looping
Glimpse	Text information retrieval tool	40M	Sequential, irregular
Sort	Unix sort utility	17M	Sequential

Table 1. Characteristics of the applications

5.2 Experiments Results

We experimented to decide the size of IR. Table2 shows the elapsed time of Dinero, Glimpse, Cscope and Sort as we vary the size of IR and set DR to 6MB.

App\IR size	0 block	1 block	2 block	3 block	4 block	5 block
Dinero	20.719	20.12	16.204	15.204	12.109	12.109
Cscope	14.1644	13.712	12.325	11.325	8.509	8.509
Glimpse	30.1005	29.833	27.633	25.633	25.49	25.49
Sort	19.967	19.123	17.83	16.83	15.855	15.885

Table 2. The elapsed time(second) of LMRU with varying the size of IR.

Table2 shows the elapsed time of LMRU does not improve even if the size of IR is over 4 blocks. The reason is that there can appear three levels of indirect blocks before reaching the data block in Linux. When we set the size of IR to 0 block, the elapsed time is same to MRU. That is to say when the size of IR is 0, LMRU works as MRU. Based on the above experiments, we set the size of IR to 4 blocks. The sum of the size of IR and DR is the size of buffer cache. Since the size of IR is very small compared to DR, the size of buffer cache is nearly the size of DR.

Figure 4 shows the effects of varying the size of buffer cache on the number of elapsed time under different buffer replacement. When we use LRU we can observe that the elapsed time of Dinero, Cscope and Sort has not changed even if we vary the size of buffer cache. That is to say, for large sequential or looping reference pattern applications are not likely to benefit from LRU although buffer cache size is increasing. But in MRU and LMRU, the elapsed time of the application is decreasing as the size of buffer cache is increasing because the resident block of the buffer cache may hit. In the case of Glimpse, since it has an irregular and sequential reference pattern, when we apply LRU, MRU and LMRU to it, the elapsed time of the application is decreasing as we increase the size of buffer cache. As figure 4 shows, the performance of MRU is low compared to LRU and LMRU. The reason is that the indirect block is fetched and replaced frequently.

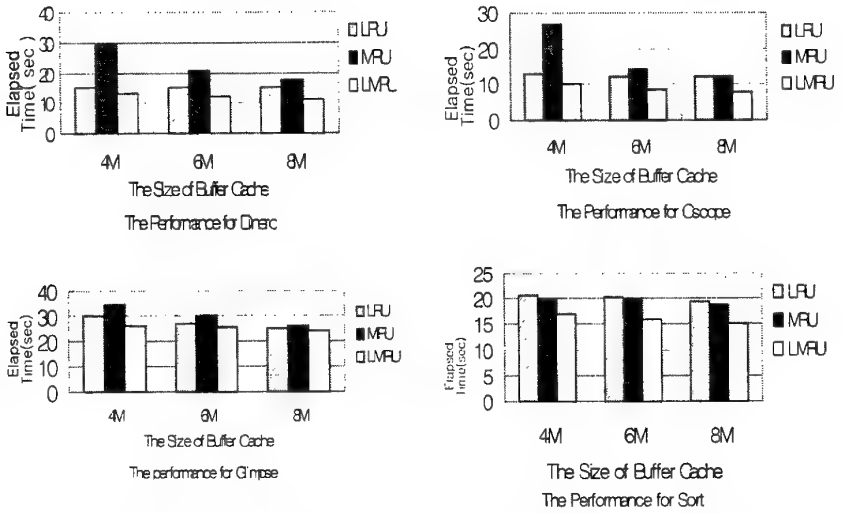


Fig. 4. Performance for various applications as changing buffer cache size and buffer replacement policies

In LMRU, since the replacement policy has a structure to maintain an indirect block and manage data block as MRU, for sequential and looping applications, LMRU outperformed LRU and MRU.

5.3 The Analysis and Experiments on the System Performance by Indirect Block

As we've shown in figure 4 of section 5.2, when we apply MRU to sequential or looping reference pattern application, it is very important to maintain indirect block so as not to degrade system performance. In this section, we present the experimentation and analysis on the influence of indirect block to the system performance.

Figure 5 shows the influence of indirect block to the system on the various applications. We used the hit ratio as a performance metrics. Since the referenced blocks to a buffer cache include not only data blocks but also indirect blocks, we represent the hit ratio as equation (4).

$$\text{Hit ratio} = \frac{\text{The sum of hit data blocks and indirect bocks}}{\text{The sum of requested data blocks and indirect bocks}} \quad (4)$$

In LMRU, the number of hit indirect blocks is represented as equation (5)

$$\begin{aligned}\text{The number of hit indirect block} &= \text{The number of requested indirect blocks} \\ &\quad - \text{The number of cold miss of indirect blocks}\end{aligned} \quad (5)$$

The effect of indirect block to the whole system is a ratio of the sum of application requested data block and indirect block to the number of hits on indirect block. We can drive the equation (6) using equation (4) and (5). Equation (6) shows an influence of indirect block to the system

An influence of indirect block on the system (6)

$$= \frac{\text{The number of hits on indirect block}}{\text{The sum of requsted block and indirect block}}$$

Using equation (6) we have tested the influence of indirect block on the Linux kernel.

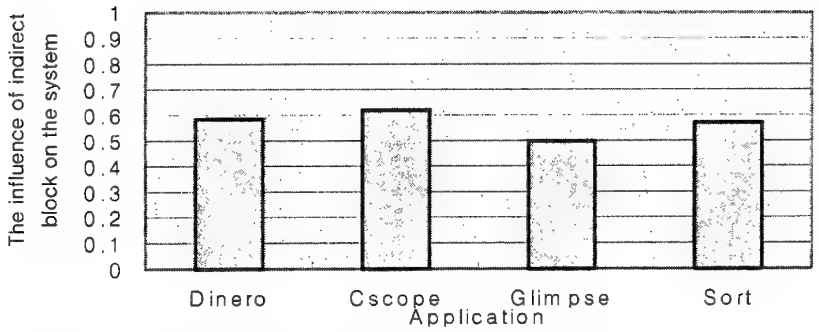


Fig. 5. Influence of indirect block on the system performances

We can observe that indirect blocks influence upon the system performance by 49.8% up to 62%. Sequential or looping reference pattern application has much influence on the system performance compared to irregular reference pattern application.

We can analyze the influence of indirect block on the system. Using equation (3) and (6) we can represent an effect of indirect block on the system as equation (7)(8) and (9). Equations can be classified by three cases according to N.

Case 1. $D < N \leq (D+P)$

$$\frac{N-D+1}{2N-D} \tag{7}$$

Case 2. $(D+P) < N \leq (D+P+P^2)$

$$\frac{2N-P-2D-2-\left\lceil \frac{N-(D+P)}{P} \right\rceil}{3N-P-2D} \tag{8}$$

Case 3. $(D+P+P^2) < N \leq (D+P+P^2+P^3)$

$$\frac{3N-2P^2-3P-3D-3-\left\lceil \frac{N-(D+P+P^2)}{P} \right\rceil-\left\lceil \frac{N-(D+P+P^2)}{P^2} \right\rceil}{4N-2P^2-2P-3D} \quad (9)$$

Applications that we used are the case of equation (8). In Linux, the size of a block is 1024 bytes and size of pointer is 4 bytes. So there are 256 pointers in a block, an indirect block can point 256 data blocks. When we apply these parameters to these equations, we can know that the minimum influence of indirect block on the system is 49.7% and maximum is 64.2%. This includes the result of figure (5). So we can know that the result of figure (5) is valid and it is very important to maintain indirect block, so as not to degrade system performance, when we apply MRU as a buffer cache management scheme.

6. Conclusion

This paper presents the results of a buffer cache management scheme for large sequential, looping and irregular patterns. When we apply MRU to large sequential or looping data it needs a structure to maintain indirect block so as not to fetch and replace frequently. To solve the problem of MRU we have designed LMRU to cache indirect block and manage data blocks as in MRU. LMRU can maintain indirect block in the buffer cache using only four blocks. LMRU improves the response time by up to 70% compared to LRU and 163% to MRU. LMRU can be used with application controlled file caching, Open Implementation method and DEAR scheme[1][2][12].

References

1. P.Cao, E.W.Felten, and K.Li. Application-Controlled File Caching Policies. In Proceedings of the USENIX Summer 1994 Technical Conference.
2. J.Choi, S.H.Noh, S.L.Min, Y.Cho. An Implementation Study of a Detection-based Adaptive Block Replacement Scheme. In Proceedings of 1999 USENIX Annual Technical Conference, June 6-11, 1999.
3. G.Glass, P.Cao. Adaptive Page Replacement Based on Memory Reference Behavior. In Proceedings of SIGMETRICS'97, pp.115-126, June 1997.
4. C.Faloutsos, R.Ng and T.Sellis. *Flexible and Adaptable Buffer Management Techniques for Database Management Systems.* In IEEE Transactions on Computers, 44(4): 546-560,April 1995.
5. P.J.Sheony,P.Goyal,S.S.Rao, and H.M.Vin. Simpony: An Integrated Multimedia File System. In Proceedings of Multimedia Computing and Networking (MMCN) Conference, pp 124-138,1998.
6. Barbara K. Pasquale and George C.Polyzos. A Static Analysis of I/O Characteristics of Scientific Applications in a Production Workload. In Proceedings of Supercomputing '93, pp 388-397,1993.
7. A.Dan , P.S.Yu, and J-Y Chung. Characterization of Database Access Pattern for Analytic Prediction of Buffer Hit Probability. VLDB Journel,4(1):127-154, January 1995.

8. M.J.Bach. The Design of the UNIX Operating System.Prentice-Hall, 1986.
9. M.Beck,H.Bohme,M.Dziadzka,U.Kunitz,R.Magnus,D.VerWorner. *Linux Kernel Internals*, Addison Wesley, 1997.
10. A.J.Smith. *Disk cache-miss ratio analysis and design considerations*, ACM Transactions on Computer Systems, 3(3):161-203, August 1985
11. B.Ozden, R.Rastogi,A.Siberschatz Buffer Replacement Algorithms for Multimedia Storage Systems. In Proceedings of IEEE International Conference on Multimedia Computing Systems 1996
12. Gregor Kiczales, "Beyond the Black Box: Open Implementation", IEEE Software Engineering, 13(1):8-11,1996

Parallel Architecture for Natural Language Processing

Ricardo Annes
FACI – PUCRS Campus II
BR 472 Km 7 - Sala 24
97.510-460 - Uruguaiana –RS – Brasil
phone 55 55 413 15 15 fax 55 55 413 12 80
annes@pucrs.campus2.br

ABSTRACT

This work presents a study on the applicability of the Multiagent paradigm to the part-of-speech tagging problem. The work is related to Computational Linguistics and Distributed Artificial Intelligence in order to propose a non sequential approach to the part-of-speech tagging problem. Natural Language Processing (NLP), and more specifically Corpus-based Processing, study linguistic phenomena under the point of view of Computer Science. NLP has given a significant contribution to man-machine communication. Multiagents Systems (MAS) is a well-established concept in the area of Distributed Artificial Intelligence. MAS's has been thoroughly studied towards application in NLP. In this work we propose a distributed architecture in which every agents acts on a specific style corpus. The proposed architecture has been implemented and it is being currently tested with an initial set of texts.

1 Introduction

The actual phase of Computer Science development challenges us to diminish the interaction problems between computers and users. The Artificial Intelligence and the Natural Language Processing (NLP) are source of research looking for different ways of minimising these problems.

The Processing based on Corpus is an approach which uses heuristics for the abstraction of linguistic knowledge and also uses stochastic techniques on large volumes of textual data (written or spoken) available for computer science processing, this makes learning by means of natural examples possible on the target language.

The treatment given to the textual data basis which supports the heap of the Human knowledge, such as those expressed in natural language, the information recovered, the Man-Machine interfaces, automatic translations, language comprehension and generation, are determinants to the NLP evolution.

This work has the main objective of presenting an investigation made about the use of Multiagent model on the Natural Language Processing through a distributed architecture of tag agents which can maximise the results of sequential approaches.

The linguistic knowledge taken from a corpus can be: specific to this corpus, generic to a group which corpora with texts of similar origin or of validity to all the language *studied/treated*. The processing of various corpora can give information about the specificity/generality levels of parts of speech.

Our proposal is an architecture of multiagent system to tag texts where multiple tag agents exchange information during the training and tagging phases. This architecture is made by a set of specialised tag on specific domains (or text styles) and of one generic tag in order to avoid redundancies.

The expected advantages of this approach are: increase of precision of tagging with a minor set of *examples* for the training according to the focus on specific domains of texts styles and a better performance on training according to the use of parallelism.

On section 2 we present the basic concepts of the three areas studied: Natural Language Processing, Processing based on corpus and the problem of corpus tag. On section 3 we present the Architecture of our system, the proposed agents model and the co-operation layer between the society agents. On section 4 we present the aspects of the system prototype implementation proposed and on section 5 we present our conclusion and proposes for future works.

2 BASIC CONCEPTS

2.1 Natural Language Processing

The Natural Language Processing (NLP) [ALLE94], Artificial Intelligence sub-area, means two areas with different focus and similar problems put together. Computer Science needs more natural interfaces, but on the other hand Linguistics needs computer science methods to develop its theories.

The challenges are not few, the concepts are not unified yet, but the development have been noticed. The textual data basis treatment which supports the heap of Human knowledge such as those expressed in natural language, the information recovered, the Man-Machine interfaces, automatic translations, language comprehension and generation, are determinants to the NLP evolution.

We must mention, at this point, that NLP treats language only as a symbolic language representation, which involves more meaningful factors such as: habits, culture, gestures, knowledge and believes.

Although there are many successful works on different areas of NLP (automatic translation, texts correction, bibliographic consults, representation formalisms, etc.) interpretation and automatic processing of knowledge available on natural language still poses problems without a complete answer whenever we try to apply to all systems a meaningful and genetic form.

The high level of inter-relationship among the linguistics domain and the tentative of manipulating natural language on an unrestricted and unlimited form contribute to the complexity of this area.

2.2 Corpus-based processing

A corpus is a collection of patterns (pieces, fragments) language which are selected and ordered according to a explicit linguistic criteria for being used as a sample of language. The linguistic criteria can be extern (when related to the participants, the occasion, the social class or the communicative function of language patterns) or intern (when related to the occurrence of standard language inside the language patterns.

The use of corpus is made for the study of a language by the knowledge of the samples used naturally on the target language.

The advantages of the corpora are: accessibility, velocity and exactness/fidelity.

The corpora can basically be plain or annotated. On the plain corpus there is no information about the text, on the annotated, we could add information such as: lexical category, syntactic structure, speech information, etc. The annotated corpora can also be called tagged.

It has been used the heuristic employment for the choice of alternatives to the syntactic analysis. One of these approaches, corpus-based processing, uses techniques based on the Probability Theory.

The approaches based on statistics were also important because it promotes the ability of effective parameters learning from the corpus processing. These algorithms start with an initial estimate of probabilities and after then the corpus is processed in order to calculate the better estimate, repeating the process until no option of a better answer could be given. This technique guarantees the convergence, although it could not discover an excellent value.

The approach basically based on knowledge emulate knowledge of human speech using techniques which comes from Specialist Systems. Systems based exclusively on rules have got a limited success. Many of the well succeeded systems use the stochastic approach. [CHUR93]

2.3 Tagging

Based on a great amount of marked texts (corpus) [CHUR93;CHAR93;MARC93] may efforts have been made in order to create programs which execute this task (tagging).

Tagging or tagging (part-of-speech tagging) is a process used to mark parts of speech at every word on a corpus. This is a very important task on the modern Natural Language Processing and in information retrieval. Tagging is made based on the context on which the word occurred in the sentence. Some words, when considered out of context, present more than one speech part (lexical ambiguity).

If we think that a certain word can be classified with more than one tag (for example: the words **book** and **love** can be either a noun or a verb), we can abstract the concept of class of ambiguity. The class noun_verb is a set of all words which can be noun or verb.

The advantage of using this concept is that we can use the smallest number of parameters to be estimated on the model used. At the Brown corpus from 50 thousand words only 4 hundred ambiguity classes are used.

Generally tags are constructed according to two main approaches:

- Statistical methods (based on probabilistic models) and

- Rule-based models

Statistical taggers analyse the texts according to an empirical focus, independent from domain or from language, through automatic construction techniques from inferred rules based on a training corpus which do not require human supervision. The tagger acquires his knowledge based on corpura established patterns [MERI94].

The taggers are created by linguists according to linguistics rulers[BRIL93; VILL95] based on language models and syntagmas, creating a grammar.

This is a manual work which depends on the domain of a specific language what makes it an expensive work. Generally a symbolic program based on rules is used to construct the tagger.

A mixed approach [BRIL92] [BRIL93] describes the tagger based on rules taken from statistical processes. The system is composed by two taggers, the first tagger creates a dictionary with the most probable category for each word, without considering sentential context . For unknown words there is a high level set of rules, such as "words starting with capital letters are common nouns", "words ending on 'ing' are verbs". The second tagger infers rules automatically from the training corpus marked context comparing the tagged corpus with the results of the first tagger.

In both approaches, the size of the training corpus must be quite large to achieve a reasonable precision.

This precision is calculated by two different ways:

- quantity of correctly tagged words divided by the total quantity of corpus words or

- quantity of correctly tagged sentences divided by the total corpus sentences.

We can notice that the second criteria is much more rigid than the first one.

The statistical tagger work can be divided into 3 stages: training, test (validation) and tagging.

During the training the system 'learns' with the tagged corpus. Precision of statistical taggers on the tagged texts is proportional to the number of entry examples during the training stage (tagged corpus) and dependent to the corpus type. Whenever the entry increases the amount of information on the tagger also increases what promotes less efficiency.

The formalism generally used on the implementation of the training phase is the HMM (hidden Markov Model) [CHAR93]. This model is a machine of finite state which makes possible to regulate transitions among states and to control the emission of the exit signal. This model presents the advantage of executing learning in an automatic way, independent of *domain*, set of tags or language.

The n-grams [MERI94] uses the concept of context or neighbourhood to solve the ambiguity problem. The model n-gram defines that we should analyse n-1 neighbour word for each word in a sentence. Generally the most used models are bigrams which analyse the precedent word and trigrams which analyse the two precedent words. The more word analysed the most precise the results and higher the cost of processing.

During the training process the tagger receives the tagged corpus and estimates the HMM parameters through algorithms of Relative Frequency and Forward-Backward [CHAR93].

After constructing the HMM and estimating its parameters, the tagger is ready to execute the tagging of words and sentences. During the test stage texts with known tagging are processed in order to analyse the results and if necessary to make the appropriate adjustments.

During the tagging process the Viterbi algorithm is used in order to discover the most probable sequence of tags for a certain sentence. [CHAR93].

3 PROPOSED ARCHITECTURE

3.1 Multiagent approach motivation

The linguistic knowledge abstracted from a corpus can be: specific to this corpus, generic to a group of corpora with texts from similar origin or of validity to all treated language. Various corpora processing can bring information about the levels of specificity/generality of grammatical categories abstracted.

On the approaches studied (statistical methods and based on rules), the size of the training corpus needs to be sufficiently great to present a reasonable precision.

The use of only one training corpus implies on the lose of dependable domain information – some words can present specific uses depending on the text type or style.

Such considerations motivate the application of distributed processing on corpora of different text style. A natural form of making this application possible is the use fo Multiagent Systems [WERN92].

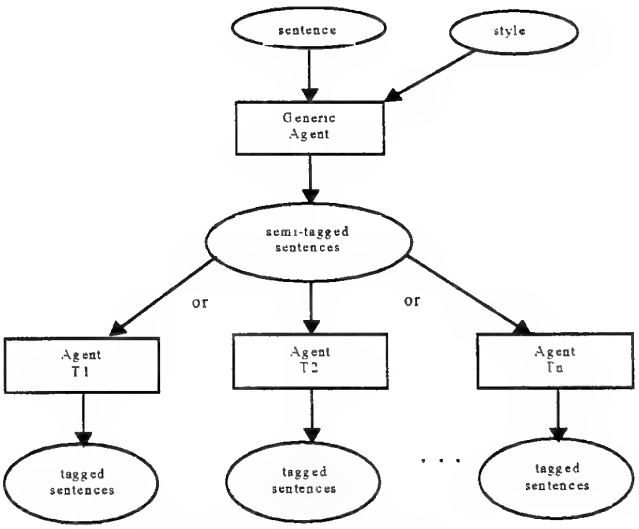
So, we propose an architecture of multiagent system to tag texts, where multiple tagger agents exchange information during the training and tagging stages. This architecture is composed by a set of taggers specialised on specific domains (or text styles) and a generic tagger to avoid redundancies:

The advantages expected from this approach is: a) tagging precision increase with a set of less examples for training; b) higher processing velocity during the training stage because of the architecture parallelism and c) better tagging performance because of the models separation as we can see on the next sections.

3.2 The society architecture

The society architecture is made by a generic agent and a set of specific agents [ANNE98]. The process is divided into two stages: Training and Tagging. During the Training stage the system acquires, through the corpora of different styles, the necessary knowledge to the Tagging stage. The Tagging stage is less complex than the Training stage as we can see below.

During the tagging stage (picture 3.1) the sentence to be tagged and its text style are passed to the generic text style). After this stage the sentence can be sent to the correspondent agent of a text style and the tagging will be completed. agent which makes a preliminary tagging, according to its knowledge (morph-syntactic categories common to any



Picture 3.1 Tagging system stage

3.3 Training stage

During the training stage each specific agent takes as entry a tagged corpus in order to acquire knowledge about the morph-syntactic category of each word of these corpora.

The agents knowledge is moulded on HMM and the protocols of communication between them had been formalised in a language based on KQML (section 4).

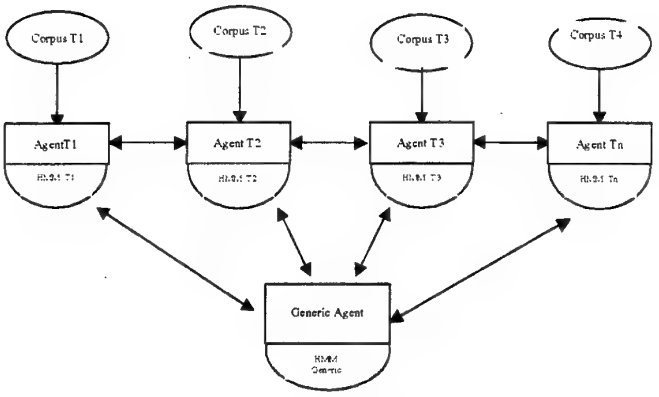
During the system training stage (picture 3.2), the agents acquire their knowledge to apply on the tagging stage. To each specific agent one corpus of a certain text style is submitted. On a co-operative way, interacting with the other society agents, each agent creates its HMM using only specific linguistic patterns of the processing style. It is the generic agent work to create HMM with common patterns to all style texts taken into account.

3.4 Agents model

According to Demazeau's generic agent description [DEMA90], the agents of our architecture present the following characteristics:

The agent's **knowledge** is represented by Markov's Models which are acquired, mainly at the training stage, from the specific corpora and the generic agent is acquired through the interaction with the other agents.

It has been chosen the Statistical Tagging developed by New Lisbon University (NLU) described by [VILL95], that was made agent to become a society member through a **co-operation layer** which makes possible to exchange information at the training stage.



Picture 3.2 System Training Stage

NLU statistical tagging is basically compounded by three modules: the **classifier**, the **HMM constructor** and the **Viterbi** module.

The **classifier** module, based on a training corpus previously tagged, creates a dictionary and a file which contains the ambiguity classes and the existent tags. The dictionary puts the training corpus words together with all the grammatical categories associated to them.

Up to this module the system does not work with words anymore, but works with the sequences which were formed by the words ambiguity classes.

The **HMM constructor** creates Markov's model using the bigrams and relative frequencies based on the ambiguity classes created by the classifier.

This way, the model is made by the sequences received from the **classification** module (ambiguity tags and classes) added the chosen tag from those listed on the respective class of ambiguity and from the estimate probability for this tag.

The estimate probability is calculated using the algorithm of Relative Frequency substituting the words by their ambiguity classes and using the contextual probability.

$p(W,T) = \Pi\, p(t_i.c_i \mid t_{i-1})$

Where:

W is the word sequence,
T is the tag sequence,
c_i is the ambiguity class,
t_i is the tag word and
t_{i-1} is the previous word tag.

The probability formula is given for:

$$p(t_i.c_i \mid t_{i-1}) = \frac{f(t_{i-1}, t_i.c_i)}{f(t_{i-1})}$$

where:

c_i is the ambiguity class,

t_i is the tag word and
t_{i-1} is the previous word tag.

Viterbi module discovers which is the most probable tag sequence of a corpus from HMM model generated by the constructor.

3.5 Co-operation layer

Aiming to accomplish the communication tasks, social reasoning and control, agent tagger is "involved" in a module which we call **Co-operation layer**.

The **Co-opweration layer** give autonomy capability to the agents and the necessary mechanisms to the accomplishment of the co-operative and the team-work tasks. This layer implements the negotiation interaction and the messages exchange between the society agents on the training and system tag stages.

Taking decision process

Through this process the society takes the decision of making the sequence generic or of keeping it specific to each agent.

If all the specific agents send the same sequence to the Generic Agent, it can calculate an estimate average probability, store this sequence and send to the Specific Agents a message asking them to eliminate from their models the sequence which now is a generic one. The estimate average probability is calculated on the weighted mean way in relation to the quantity of words of each training corpus. This way the biggest corpora will be represented on a more relevant way on this average.

During the tagging stage, the generic agent uses the estimate average probability in order to pre-tag the texts received before sending to the correspondent specific agent the text style which will complete the tag.

At the moment the specific agent get to the end of their corpus data, these ones send messages to the Generic Agent telling about this event.

After receiving the message of end of data from all Specific Agents, the Generic Agent sends to them message of end of process.

4 PROTOTYPE IMPLEMENTATION

The prototype implementation was developed on C++ language for Unix machines with Solaris operational system at MASENV environment which uses the object classes of DPSK+P.

The DSPK+P [CARD92] is a library of classes which uses a general purpose data structure in order to construct shared objects (classes and instances). This data structure is compound by a class name (string) plus a facet set (slots), with pairs of value-attribute and defined by class C++ DPSK_OBJECT. This class provides the basic construction for the interface C++ of DPSK+P.

The structural base of interface C++ is compound by: Slots which form the fundament to local objects, local objects form the fundament to shared objects and lockers (transactions) which support classes and recoverable instances.

The MASENV (Multi Agent Sofware ENVironment) corresponds to a layer which goes on DPSK+P which allows to generate the communication

through messages among different processes called *agents*. These agents are connected among themselves so that they can form a net called society.

4.1 Training Corpora

On the architecture evaluation three training corpora were used, each one with a type of text: academic style, sportive style and religious stype.

The corpora were submitted to a statistical analysis with the objective of showing the differences on the tag sequences found in each text style. On table 4.7 we present a summary in order to demonstrate that certain tag sequences have similar probability in the three corpora while others differs a lot, this shows that certain sequences are generic to the corpus used and others are specific to certain text style.

TAG	Academic	Sportive	Biblic
Noun	30% PREP	29% PREP	**33% CONJ**
Verb	**23% ARTI**	26% PREP	24% PREP
Pronoun	**46% NOUN**	43% VERB	41% VERB
Preposition	57% NOUN	39% NOUN	44% NOUN
Conjunction	**31% ARTI**	29% VERB	28% VERB
Article	86% NOUN	82% NOUN	60% NOUN

Table 4.7 - Tag Sequences

5 CONCLUSION AND FUTURE WORK

This work proposes a distributed architecture for a corpora tag through the Multiagent paradigm.

We are in need of great volumes of tag corpora which is fundamental to the Natural Language Processing and much have been done to achieve this.

In the last few years, works which apply the Multiagent paradigm have emerged in different areas and they have got success on the Natural Language Processing.

Based on this we have studied the Multiagent model, Natural Language Process in general and Processing based on Corpus and taggers in special.

Our proposal wants to contribute not only with the Natural Language Processing, in relation to the efficiency and to the tag problem focusing, but also to the Multiagent systems by presenting a proposal of distributed architecture and tagger agents.

This work is the result of a proposal of implemented Multiagent artchitecture and on tests with an initial set of texts.

This is a generic architecture which can be used to tag text of other languages besides Portuguese.

The approach used is also independent of the tagger used, so it can be applied to other statistical taggers. The implementation model allows the use of the agents' co-operation layer and also to use another tagger if adapting the code.

One of the system limitation is the impossibility of negotiation among agents to take place simultaneously to the learning of each corpus. It was not possible because of the tagger characteristics. The implementation of this characteristic would make the system more complex, once there would be the necessity of synchronisation, even though, we believe that there could have been some profit on performance.

The characteristics of Mental State of the agents were not formally model in this work. We believe that being the negotiation process so simple, such characteristics would not be necessary.

After this work, we intend to make more tests with the developed prototype in order to get means to compare the architecture performance in relation to tagger agent, as well as with other taggers. The comparison measure should be not only in terms of tagging efficiency (correctly tagged words divided by the total of words), but also in terms of computer science resources used.

Our future work will be, for sure, to improve the implemented prototype. To recognise the text style to be tagged is a desirable characteristic of the system.

Improve the negotiation process and make it simultaneous to the construction process of each specific agent model is the type of work which we believe can make our approach richer.

6. REFERENCES

[ALLE94] ALLEN, J. **Natural language understanding**. The Benjamin/Cumming Company. 654p. 1994.

[ANNE98] ANNES, R.; OLIVEIRA, F. M. **Multi-Agent Part-of-Speech Tagging**. Multi Agent Systems Models Architecture and Applications II Iberoamerican Workshop on D.A.I. and M.A.S. Francisco J. Garijo and Christian Lemaitre (Eds.) Toledo,Spain, 1998.

[BRIL92] Brill, Eric. **A simple rule-based part of speech tagger**. In proceedings of The DARPA speech and Natural Language Workshop. p.112-116, 1992.

[BRIL93] BRILL, E,; **A Corpus-Based Approach to Language Learning**. Tese de Doutorado. Universidade da Pensilvania, 154p. 1993.

[CARD92] CARDOZO, E. **DPSK+P User´s Manual C++ Interface Version 1.0**. Engineering Design Reaserch Center, Carnegie Mellon University, 1992.

[CHAR93] CHARNIAK. E. **Statistical language learning**. London: Abradford Book.The MIT Press. 1993. 170p.

[CHUR93] CHURCH, K.; MERCER, R. **Introduction to the Special Issue on Computational Linguistics Using Large Corpora**. Computational Linguistics Vol. 19 N.1. 1993.

[DEMA90] DEMAZEAU, Y.; MÜLLER, P.P. **Decentralized AI**. Morgan Kaufmann, 1990.

[MARC93] MARCUS, M.P. SANTORINI, B e MARCINKIEWICZ, M.A. **Building a Large Annotated Corpus of English - The Penn Treebank**. Computacional Linguistic Vol. 19 N.2. 1993.

[MERI94] MERIALDO, B.; **Tagging English Text with a Probabilistic Model**. In Computacional Linguistic, V20, n2 p.155-171, 1994

[VILL95] VILLAVICENCIO, A. **Avaliando um Rotulador Estatístico de Categorias Morfo-sintáticas para a Língua Portuguesa**. Dissertação (Mestrado), UFRGS, 1995.

[WERN92] WERNER, E. "**The Design of Multi-Agent Systems**" in Decentralized AI3. WERNER, E. & Demazeau, Y., (eds). Elsevier Science Publishers B.V.,1992.

A Platform Independent Parallelising Tool Based on Graph Theoretic Models

Oliver Sinnen* and Leonel Sousa

Instituto Superior Técnico - INESC
Rua Alves Redol 9, 2
P-1000 Lisboa, Portugal
{oliver.sinnen, las}@inesc.pt

Abstract. Parallel programming demands, in contrast to sequential programming, sub-task identification, dependence analysis and task-to-processor assignment. This paper presents a new parallelising tool that supports the programmer in these early challenging phases of parallel programming. In contrast to existing parallelising tools, the proposed parallelising tool is based on a new graph theoretic model, called annotated hierarchical graph, that integrates the commonly used graph theoretic models for parallel computing. Part of the parallelising tool is an object oriented framework for the development of scheduling and mapping algorithms, which allows to rapidly adapt and implement new algorithms. The tool achieves platform independence by relying on internal structures that are not bound to any architecture and by implementing the tool in Java.

1 Introduction

The programming of a parallel system for its efficient utilisation is complex and time consuming, despite the many years of research in this area. Parallel programming demands much more than sequential programming, since (i) sub-tasks which can be executed in parallel must be identified, (ii) the dependence between the sub-tasks has to be analysed and (iii) the tasks have to be mapped and scheduled to a target system.

Due to the high complexity of parallel programming, research on methods, mechanisms and tools to support the parallelisation of tasks has been encouraged. Many tools, languages and libraries emerged from this research. Even tough, the majority of them only supports the coding and result analysis, there are some tools, in this paper called parallelising tools, that address the challenging early phases of the development of a parallel program (e.g. CASCH [1], Task Grapher [2], PYRROS [3], CODE [4], Meander [5], or [6]).

Parallelising tools use commonly graph theoretic models for the representation of a program to be parallelised. Computation is associated with the vertices and communication with the edges of the graph. The graph is generated from an initial description of the program to be parallelised (for example in a proprietary task graph language [3]) or it is interactively constructed in the environment of a parallelising tool [4]. The Directed Acyclic Graph (DAG) [7] is the most common model employed and a lot of recent research has been accomplished in the area of mapping and scheduling algorithms (e.g. [8, 9]) for this graph model. A shortcoming of this graph model is, however, its incapability to model code cycles explicitly. Loops with a large or even, at compile time, unknown number of iterations cannot be represented with a DAG. Other graph models, like the Temporal Communication Graph (TCG) [10] or the Interactive Task Graph (ITG) [11] overcome this limitation.

While the majority of the parallelising tools is based on only one graph model, which is mostly the DAG (e.g. [1–3]), they also often use only a limited number of scheduling and mapping algorithms. This further limits their area of application, since the scheduling and mapping heuristics often have affinity for certain types of applications and parallel architectures. In other words, a

* Candidate to the Best Student Paper Award

scheduling heuristic may produce good results only for some types of programs on certain architectures. An extensible and adaptable design of the parallelising tool is necessary to include algorithms tailored for different types of applications and parallel architectures.

A myriad of scheduling heuristics for the DAG model [12–15] has been developed by the parallel computing community. Scheduling heuristics assume in general a homogenous multiprocessor system with fully connected processors. This characteristic is, however, rarely found in a real world target machine. Only a few algorithms were proposed that take a realistic hardware architecture into account [2, 16, 17]. One should expect better scheduling results for real world machines from these algorithms, but unfortunately no practical comparison of scheduling heuristics has been published. The comparisons found in literature [12–15] use the schedule length as a measure, but not the actual execution time on a target system. For a parallelising tool to benefit from the programmers knowledge about the target system, it must provide a scheme for the specification of these characteristics in a way that the appropriate algorithms can use them.

A parallelising tool can benefit from the natural strength of humans, such as information abstraction, pattern matching and problem decomposition, when it leaves some of the parallelisation decisions to the user. These decisions can be, for example, the choice of a scheduling algorithm, the characterisation of the target machine or the manipulation of the parallel program structure. A visual environment, which displays the parallel structure (in two or three dimensions) and allows interactive manipulations [4, 6] gives the user the most flexibility. In such an environment, the programmer can understand, correct and optimise the parallel structure.

This paper proposes a parallelising tool that addresses the above discussed issues. The parallelising tool is based on a new hierarchical graph model that integrates multiple graph theoretic models, to support a wide range of applications and parallel architectures. Its design allows the adaption and modular extension of parallelising algorithms, by integrating new algorithms for scheduling, mapping and structure manipulation. A visual 3D environment displays the parallel structure of the program, to support the parallelisation decisions of the programmer. For platform independence, the parallelising tool is implemented in Java.

The rest of this paper is organised as follows: Section 2 presents an overview of the parallelising tool and introduces its principal components. After the classification and comparison of graph theoretic models, a new graph model called annotated hierarchical graph is presented in Section 3. The subsequent sections then discuss the components of the tool. Section 4 presents the type of inputs accepted by the tool, Section 5 describes the interactive visual environment, Section 6 analyses the module block that provides the algorithms, and Section 7 provides information about the output of the parallelising tool. We finally conclude this article in Section 8.

2 The Proposed Parallelising Tool

The overall structure of the parallelising tool we are currently developing is shown in figure 1. As indicated by the shaded areas in the figure, the tool is divided into four main parts. The input part serves for the initial description of the task to be parallelised. Modules within this part generate an annotated hierarchical graph from a task defined, for example, in a sequential programming language (e.g. C) or as an equation (e.g. in LaTeX syntax). The annotated hierarchical graph is further on the representation of the task to be parallelised and forms the core of the parallelising tool. In the central part, this graph is visualised in a 3D graphical environment. This environment allows the analysis and manipulation of the task's parallel structure and, in addition, the possibility to interactively construct an annotated hierarchical graph in a direct way. To provide the algorithms and methods for parallelisation, such as structure manipulation, scheduling and mapping, the central part interfaces to a module block containing a pool of algorithms. Taking the annotated hierarchical graph as input, these algorithms can be executed by the user for mapping and scheduling a task onto a target machine. Algorithms that take the target machine's characteristics into account, receive this information as an additional input, provided by the user. The mapping and the schedule of the task is the principal output of the tool. A programmer can then use traditional tools to code the task, guided by the obtained schedule and mapping. A long

term objective in the development of the tool is to employ code generators which automatically build the program.

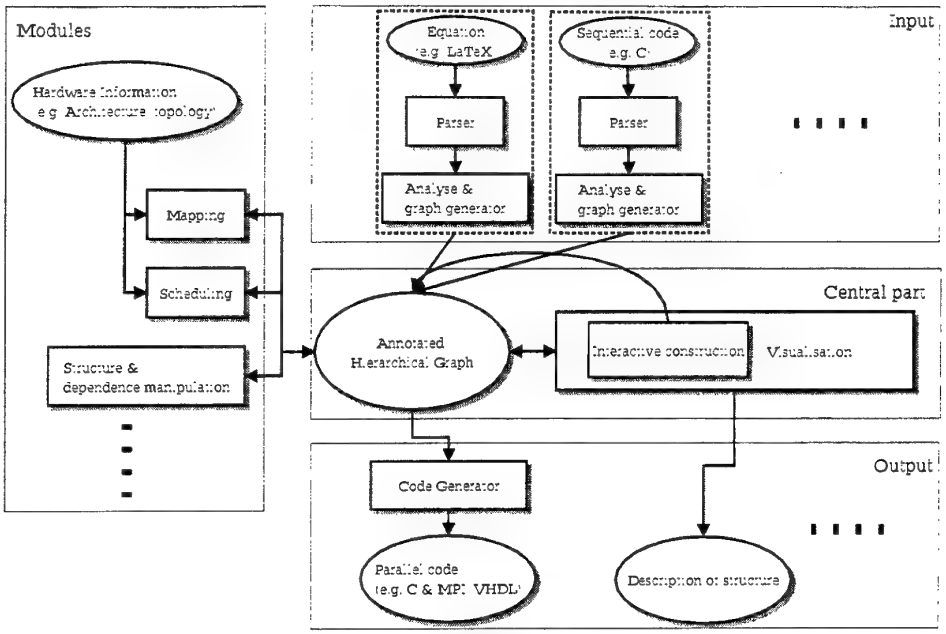


Fig. 1. Overview of the parallelising tool

3 Annotated Hierarchical Graph

The annotated hierarchical graph forms the core of the proposed parallelising tool. Unlike other parallelising tools, our tool is not based on only one graph theoretic model. It rather tries to integrate various graph models and thus to benefit from their combined advantages. Before the annotated hierarchical graph is described, we, therefore, analyse and classify the utilised graph theoretic models.

3.1 Graph Theoretic Models

In a graph theoretic abstraction, a parallel program consists of two activities: computation and communication. Computations or tasks are associated to the nodes and communications to the edges of the graph. All instructions of one task are executed in sequential order, i.e. there is no parallelism within one task. A node can begin execution only when all inputs have arrived and outputs are available at the end of the task's execution.

In [18] a classification scheme for graph theoretic models was proposed. In the context of a parallelising tool, a graph model is distinguished according to (i) the parallel computations that can be modelled, (ii) the supported parallel architectures and (iii) the available algorithms.

Within these three classification groups the models are analysed according to (a more detail description of the classification scheme is in [19, 18]):

1. Parallel computations
   - Granularity - fine grained, medium grained and coarse grained

- Iterative computations - how are iterative computations modelled?
- Regularity - can the model represent regularity explicitly?

2. Parallel architectures
   - Data and instruction stream - SIMD and MIMD streams
   - Memory architecture - shared memory (UMA), distributed memory and shared distributed (NUMA) memory
   - VLSI systems - VLSI array processors with synchronous data and control flow
3. Proposed algorithms
   - Dependence analysis and exploitation of parallelism
   - Task-to-processor mapping
   - Scheduling

To choose a graph theoretic model for the parallelising tool, we have analysed and compared the common graph theoretic models using the above classification scheme [19]. From the many existing models, DAG, ITG, and TCG, turned out to be of interest for the parallelising tool. In the following sections these three models are briefly discussed and the aspects which had influence on the annotated hierarchical graph structure are pointed out.

Directed Acyclic Graph (DAG) The designation DAG [7] merely reflects the graph theoretic nature of this graph model, which is consequently directed and acyclic. Interesting for a parallelising tool are node and edge weighted DAGs, as they well reflect parallel computations with non-uniform computation and communication costs.

The acyclic property of the DAG imposes a restriction on how parallel computations are modelled. Iterative computations, which build a cyclic structure, are urged to be modelled in a certain form. A coarse-grained approach consists in projecting the iterative part of a parallel computation onto one task. On the other hand, in a fine-grained approach only the tasks of one iteration are modelled, without taking into account the inter-iteration dependence. For a complete fine-grained representation, the iterative computation may be 'unrolled' where each iteration is represented by its own sub-graph, and these sub-graphs are connected according to the inter-iteration dependence.

In the last approach, the size of the DAG increases linearly with the number of iterations. In practice, this representation may generally be used only for small numbers of iterations. Also, the number of iterations has to be known at compile time; iterative computations whose iteration number is only known at runtime cannot be modelled with this approach.

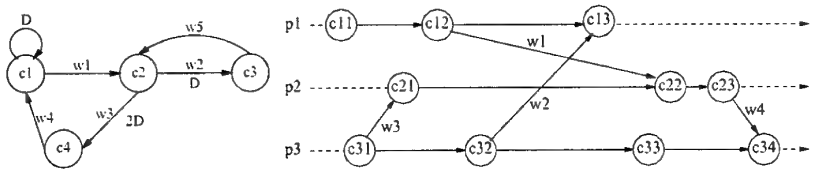
The typical granularity of modelled computations is coarse-grained, as the alternative designations as task graph and macro-dataflow graph indicate. The DAG is usually employed for mapping and scheduling on distributed-memory architectures. Node and edge weighted DAGs are only used for MIMD streams, since no spatial regularity is exploited by the DAG model, which is necessary to support SIMD streams.

A myriad of mapping and scheduling algorithms [12–15] were proposed based on node and edge weighted DAGs, whose directed and acyclic properties allow efficient scheduling algorithms.

Iterative Task Graph (ITG) The ITG [11] belongs to the group of data flow graphs, which model the flow of data or signals in a computation. Data flow graphs are directed graphs, but in contrast to the DAG model, data flow graphs can incorporate cycles and allow thus a more compact representation of computations. For a data flow graph to represent a valid computation, cycles must include at least one **delay** [20]. A delay is represented by a weight associated with an edge. It may be expressed as a multiple of a time unit (often denoted D) or the number of iterations the communication between two nodes is to be delayed. A delay "breaks" the precedence-constraint cycle and allows thus a computable representation of iterative computations. Owing to the delays, data flow graphs can model intra-iteration (without delays) and inter-iteration (with delays) dependence. The efficient scheme for representing iterative computations, reduces essentially the number of nodes in a graph compared to an unrolled DAG and allows nondeterministic

numbers of iterations. Moreover, iterations are represented in a more intuitive and structured way. Since iterative computations can be modelled in detail, the granularity is typically fine-grained to medium-grained.

The Iterative Task Graph also contains edge and node weights to reflect the computation and communication costs (figure 2a), which are mainly used for exploiting parallelism, for mapping and for scheduling. Unfolding, re-timing or software pipelining are some examples of the transformations applied to parallel computations represented by this model. The ITG is appropriate for computations with arbitrary costs on (shared) distributed memory architectures and VLSI systems, given that it explicitly provides computation and communication costs. As no regularity is included in the ITG, except for iterative computations, MIMD streams are the typical data and instruction streams supported by the model. For SIMD streams, the ITG leaks spatial regularity.



ci/cij- computation cost; wi - communication cost; iD - delay; pi - process

Fig. 2. The Iterative Task Graph (a) and the Temporal Communication Graph (a)

Temporal Communication Graph (TCG) The Temporal Communication Graph [10] is based on the space-time diagram introduced by Lamport [21]. The TCG is a directed and acyclic graph that is process and phase oriented. A computation is divided into sequential processes $p_1, p_2, \ldots, p_n$ and every node of the graph is associated with exactly one process. A node, associated with process p_i, has at least one edge pointing to its direct successor on process p_i (intra-process dependence) and may also have a communication edge to a node of another process p_j (inter-process communication). A node weight reflects the computation costs and a weight associated to an inter-process edge represents the communication costs. Communication between nodes of the same process is considered to be negligible. Figure 2b illustrates a TCG with three processes. As the graph built by the nodes and edges is a directed and acyclic graph it may be considered a node and edge weighted DAG with zero communication costs for intra-process edges.

A TCG can be described with the aid of the LaRCS [10] graph description language, which allows to specify phases of computation and communication. These phases may exploit spatial and temporal regularity, for example a loop is a phase of temporal regularity. On the one hand, this process and phase oriented perspective of parallel computations limits the flexibility of the model. On the other hand, the TCG draws its power from this scheme, as iterative and other regular computations are described in an efficient way. The process-oriented view is, moreover, intuitive to many programmers for describing computations.

In the OREGAMI tool [22] the TCG is used for mapping and scheduling. The algorithms employed use the regular structure of the TCG. Furthermore, mapping and scheduling algorithms based on the Task Interaction Graph (TIG, an undirected graph model [19]) and on the DAG model may also be used. As mentioned above, the TCG may be considered a node and edge weighted DAG and projecting the TCG along the time axis yields the TIG.

The TCG, considered as a DAG, underlies the same limitations to model an iterative computation as the DAG model itself. However, the TCG can be treated in parts due to its description in phases. In the OREGAMI tool, for example, successive portions of the graph are generated for

mapping and scheduling as needed. As a result, the complexity and the memory requirement are reduced for mapping and scheduling.

Relations Between the Graph Models Apart from the characteristics discussed above, there exist relations between the graph models, some of which were already mentioned during the above discussion. These relations can be used to transform one graph representation into another, for the purpose of applying an algorithm only available for one graph representation or to yield a more compact representation. In figure 3 the relations of the discussed graph models are depicted. Three types of transforms between graph models can be defined: reduction, projection and unrolling. By *reduction* of graph properties, a complex graph model can be transformed into a simpler model, whose properties build a subset of the complex model's properties. With *projection*, a graph model can be transformed into another model with a more compact representation of the parallel computation. The reverse process to projection is *unrolling*. The ITG, for example, is unrolled to a DAG by constructing a sub-graph for every iteration and connecting these sub-graphs according to their inter-iteration dependence.

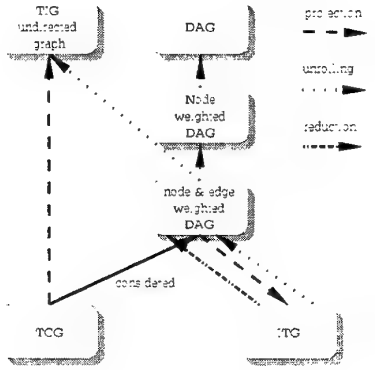


Fig. 3. Relations between graph models

3.2 Structure of the Annotated Hierarchical Graph

A conclusion drawn from the comparison of the graph theoretic models is that none of the graph models is universal. In other words, none of the models is capable of representing every type of application. DAGs are mainly used for a coarse grained representation, ITGs can only model iterative computations and TCGs are limited by its process and phase approach.

Inspired by the TCG, we developed an hierarchical model that can represent coarse grained as well as iterative computations, but which is not limited by a process-phase orientation. The principal idea is that a node of a graph can itself be again a graph, as shown in figure 4a. In this example, the directed main graph consists of the nodes n_1, n_2, n_3, of which node n_3 is itself a directed graph with the nodes n_{31}, n_{32}, n_{33}. The subgraph is cyclic and represents an iterative computation, whose dependence cyclic is broken by the delay D on the edge between node n_{33} and n_{31}.

Formally, a hierarchical graph G is a pair (V, E), where V is a finite set of vertices connected by a finite set of edges E. An element $e = (u, v)$ of E denotes an edge between the vertices $u, v \in V$. An edge (u, v) denotes an edge *from u to v* and, therefore, $(u, v) \neq (v, u)$. Note, that loops and self loops are possible. A vertex $u \in V$ can itself be a hierarchical graph G_u, where the edges entering vertex u, $(v, u) \in E, v \in V$, enter the source vertices of G_u and the edges leaving vertex u, $(u, v) \in E, v \in V$, leave the sink vertices of G_u. Source vertices are those vertices which,

after removing all edges with delay, have no entering edge and sink vertices are those which. after removing all edges with delay, have no leaving edge. In the example of figure 4a node n_{31} is a source vertex and node n_{33} is a sink vertex.

The hierarchical graph can be made a simple directed graph by substituting the nodes that are themselves graphs with their respective graphs. This is shown for the example of figure 4a in figure 4b, where node n_3 was substituted by its graph.

The hierarchical graph model permits to represent iterative and none iterative computations in one graph model in a compact form. Therfore, this graph integrates the DAG and the ITG into one model without being limited by a process and phase orientation like the TCG. Depending on the purpose, the graph can be interpreted in various ways. An algorithm may consider only the coarse grained task graph (only considering the highest hierarchical level) or the sub-graphs can be treated separately. It is also possible to expand the graph to a flat directed (cyclic) graph as done with the example from figure 4a in figure 4b. The latter can further be unrolled (supposed that the number of iterations is known for cyclic parts) to a DAG. The hierarchical graph model provides the flexibility to represent a wide range of applications and still all the algorithms proposed for the three graph models discussed above can be employed.

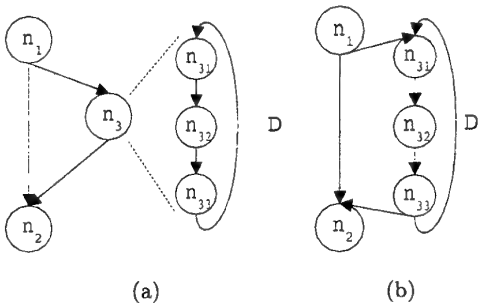


Fig. 4. Hierarchical graph (a) and expanded hierarchical graph (b)

Annotation Associated with every node of the hierarchical graph is an annotation. This annotation is in textual form and represents the computation executed by the node. Code in C or VHDL are two examples for textual annotations. The annotation may serve for the estimation of the computation time or it may be used for automatic code generation at the output of the parallelising tool. Moreover, the textual task description allows to estimate the execution costs of the task depending on the architecture of the target machine.

The communication of the edges is also described by an annotation, which represents the data structure transmitted on the edge and the amount of data. Again, this can be used to estimate the communication costs or for the code generation at the output of the tool. For example, a `send()` and a `receive()` command may be inserted, with the respective data structure as parameter, in the code of the source and sink node of the edge, respectively [1]. With the knowledge of the amount of data transmitted on the edge, the communication costs can be estimated according to "startup cost+transmission speed×amount of data" [11], considering the target machine's characteristics.

The interpretation of the annotations is left to the various algorithm and, thus, the annotated graph structure is not linked to a certain form of task or communication representation. Also, the graph representation of a program is platform independent, as costs can be estimated only when needed. Note, that a textual description of a task or communication is, of course, not obligatory, and the user can also provide estimated costs.

4 Program Input

A program to be parallelised must be initially described in an adequate form to be analysed in a parallelising tool. This description is crucial for the exploitation of parallelism. Various approaches for this initial description are used in parallelising tools. Parallelising compilers commonly use an augmented sequential programming language (e.g. HPF, pC++, Split-C) and concentrate on the parallelisation of (cost intensive) loops [23]. The CASCH parallelising tool [1] takes as input a program with procedure calls and creates a node for each procedure in a DAG representing the program. A proprietary task graph language is used by PYRROS [3] for the construction of the DAG. In the CODE programming environment [4] a subset of the C programming language is used to define the tasks computation, which is primarily used to call coarse grained functions. The structure of the DAG is constructed interactively in CODE's environment. In [6] a set of algebraic equations is entered in an iterative editor, from which a dependence graph is generated.

The core of our parallelising tool is the annotated hierarchical graph. Consequently, every initial input from which such a graph can be generated is feasible. Therefore, the proposed parallelising tool is not limited to any particular form of initial description. The input is rather modularised to allow different initial descriptions. This is even important, as the tool supports coarse grained as well as iterative computations. The above referenced parallelising tools use one initial description depending on the type of computation they support.

As shown in figure 1, we currently implement two types of initial task description. One is a description as simple algebraic equations. From certain equations found in signal processing, for example recurrence equations, it is straight forward to generate a dependence graph [20]. In contrast to [6], an equation is, however, specified in textual form (with a small subset of the LaTeX math syntax) and not interactively entered. The equation is then parsed and analysed and a hierarchical graph is generated. The functions executed by every node are specified in the textual annotations of the graph. Of advantage is the utilisation of the LaTeX syntax, since any editor that supports LaTeX can be used to specify the equation comfortably. The graph generated form an equation has typically an iterative structure.

The other type of description that generates iterative structures is the specification of loops with a simple subset of the C language. Rather than parsing a hole program, this input analyses only small code fragments consisting of nothing but a loop that is parsed and analysed for the construction of an iterative graph. For the analysis of the code, techniques found in automatic program parallelisation are employed [23]. The annotations of the graph's node consist of the code parts found in the initial description of the task.

Coarse grained computations can be specified as an interactively constructed graph in the visual environment,which is presented in the next Section.

5 Visual Environment

The central part of the parallelising tool is a visual environment (figure 1). Here, the graphs generated from the initial descriptions are visualised. When fully implemented, the graph is displayed in three dimensions and the programmer can change the viewpoint, edit annotations and manipulate the structure of the graph. Moreover, a full annotated hierarchical graph can be interactively constructed in this visual environment. It is also possible to construct the coarse grained structure of a program and to use the other input types to generate finer grained iterative computations that can be integrated in the overall structure. We are currently integrating a first experimental environment into the parallelising tool. Figure 5 shows the graph of a localised matrix-matrix multiplication (N=4) visualised in the experimental environment.

With the interactive environment, the parallelising tool can benefit from the natural strength of humans, such as information abstraction, pattern matching and problem decomposition. The programmer can take decisions, which could not, or only unsatisfactorily, be taken automatically. These decisions can be, for example, the choice of a scheduling algorithm or the manipulation of the parallel program structure. In such an environment, the programmer can understand, correct and optimise the parallel structure.

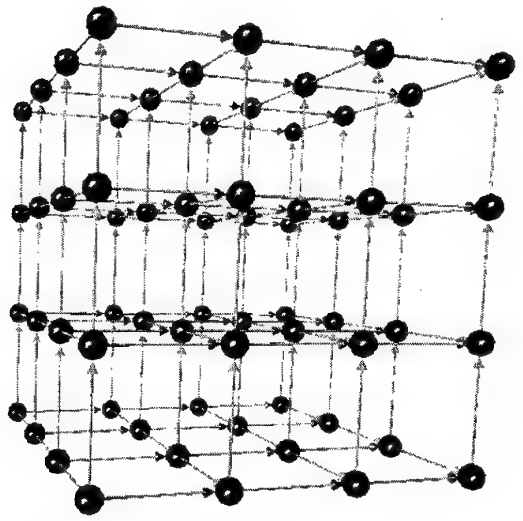


Fig. 5. Visualised graph for a matrix-matrix multiplication (N=4)

The scheduling, mapping and structure manipulation algorithms, which can be applied to a graph, can be chosen from a set of algorithms provided by the module block, to which the visual environment interfaces.

6 Module Block

The module block is responsible to provide the parallelising algorithms for our parallelising tool. It is conceived to provide a wide range of algorithms and to be adaptable and extensible for new algorithms. To achieve this goal, we developed an object oriented framework for scheduling and mapping algorithms.

The framework was implemented in Java and we employed the Collection framework of Java 2 (aka Java 1.2) for the basic data structures. It is composed of the following packages:

- `graph` - This package comprises a class hierarchy for a general graph framework. On the top of the hierarchy is a multi-graph which permits directed or undirected edges, cycles and parallel edges and the hierarchy goes down to trees and directed acyclic graphs. In conjunction with this classes, basic graph algorithms like BFS, DFS, topological order or connected components are provided.
- `hierarchicalgraph` - This package contains the classes for the annotated hierarchical graph. It is based on the `graph` package and provides the elements for the hierarchical structure and the annotations. As the hierarchical graph forms a superset of the DAG and ITG model, it can be used to represent these models.
- `schedule` - In the `schedule` package, the classes to represent mappings and schedules of programs represented by graphs are provided. This includes classes for the simple mapping of sub-tasks to processors (clusters), the definition of the relative order among the sub-tasks of the same processors, and the exact schedule of sub-tasks with their starting and finishing times. Methods for the visualisation of a schedule as a Gant chart are also included.
- `architecture` - This package is used for the definition of the target machine's architecture. It is also based on the `graph` package, as the architecture of a parallel machine is described as a graph. Generally, this is an undirected graph (however a directed graph is also possible), where the nodes represent the processors and the edges the communication links. Weights associated with the processors represent their relative processing speed and the weights of the

links their communication speed. Common architectures, as meshes, hypercubes or rings, are generated by method invocations.

The hierarchical graph class provides basic functions that are commonly used by scheduling and mapping algorithms. Examples are the calculation of the bottom or top level of a node, the critical path, or the unrolling of cycles. These functions reduce the effort for a programmer to implement a new scheduling algorithm.

As the annotated hierarchical graph is based on multiple graph models, algorithms based on different models can be employed in the parallelising tool. The most scheduling and mapping algorithm were proposed for the DAG model [12–15]. As mentioned in the introduction, only few algorithms take the target system's architecture into account. Since we expect better results from these algorithms, three of them - MH [2], DLS [16], BSA [17] - were among the first algorithms implemented for the parallelising tool. Most of the DAG algorithms analyse the structure of the DAG for scheduling. New approaches exist that take genetic algorithms into account [24, 25].

Apart from the DAG algorithms, algorithms based on the ITG are being implemented. For this graph model unfolding, re-timing and software pipelining are popular techniques [26, 27, 11]. Some of these algorithms utilise again DAG scheduling algorithms for partially unfolded ITGs.

To benefit from regular structures of graphs, especially from graphs derived from equations, techniques known from the VLSI processor design [20] are employed. These techniques use the regular structure to project (map) nodes to processors and to schedule synchronous executions.

7 Output

The output of the tool is the parallel structure of the program, the schedules and mappings. From the initial description of the program the user obtained a parallel structure displayed in the visual environment. By manipulating this structure, applying mapping and scheduling algorithms the user receives a guideline for the implementation of the program with the classical parallel tools. The user can read off the partition into subtasks from the displayed graphs as well as the dependence between these sub-tasks. The schedules displayed as Gant charts allow the programmer to define the execution order and/or starting time of the sub-tasks. The tool permits the programmer to experiment with mapping and scheduling algorithms *before* the actual implementation of the program. This is in contrast to classical parallelisation tools that display the performance of the program *after* the implementation.

A long term objective of the parallelising tool is to utilise code generators for the implementation of the program. A code generator can include communication and synchronisation primitives in the code annotated to the nodes, according to the edges of the graph [1, 4]. This can be done in a platform independent form (e.g. C with MPI functions, VHDL) or by utilising communication primitives for the specific architectures and platforms.

8 Conclusions

This paper presents a new parallelising tool based on graph theoretic models. Its design is platform independent as it is implemented in Java and as its internal structures are not bound to any architecture.

We proposed a new graph theoretic model, called the annotated hierarchical graph, that integrates the wide spread models DAG, ITG and TCG. This graph model renders the tool universal, since it is capable of representing coarse grained and iterative computations in a single model and in a compact form. Existing parallelising tools are limited in their range of supported applications. With the annotated hierarchical graph, techniques from different areas of scheduling and mapping research are integrated into one tool.

We described the elements of the parallelising tool and pointed out its modular structure. The proposed framework for the development of parallelising algorithms allows to rapidly implement and adapt new algorithm for the parallelising tool. Moreover, the tool permits to develop new

algorithms without the need for a proprietary test environment. The visual environment supports the user in parallelising decision, because the display of the parallel structures of a program help the user to understand, correct and optimise these structures.

In the future, the advantages of this new tool has to be demonstrated with the parallelisation of programs that benefit from the graph's hierarchical structure.

References

1. Ishfaq Ahmad, Yu-Kwong Kwok, Min-You Wu, and Wei Shu. Automatic parallelization and scheduling of programs on multiprocessors using CASCH. In *Proceedings of the 1997 International Conference on Parallel Processing (ICPP'97)*, pages 288–291, Bloomingdale, Illinois, USA, August 1997.
2. Hesham El-Rewini and T. G. Lewis. Scheduling parallel program tasks onto arbitray target machines. *Journal of Parallel and Distributed Computing*, 9(2):138–153, June 1990.
3. T. Yang and A. Gerasoulis. PYRROS: static scheduling and code generation for message passing multiprocessors. In *Proc. of 6th ACM International Conference on Supercomputing*, pages 428–437, Washington D.C, July 1992.
4. Rajeev Mandayam Vokkarne. Distributed execution environments for the CODE 2.0 parallel programming system. Master's thesis, University of Texas at Austin, May 1995.
5. Guido Wirtz. Developing parallel programs in a graph-based environment. In D. Trystram, editor, *Proc. Parallel Computing 93, Grenoble, France*, pages 345–352, Amsterdam, September 1993. Elsevier Science Publ., North Holland.
6. Elena V. Trichina and Juha Oinonen. Parallel program design in visual environment. In *IEEE International Conference on High Performance Computing*, Bangalore, India, December 1997.
7. V. Sarkar. *Partitionning and Scheduling Parallel Programs for Execution on Multiprocessors.* MIT Press, Cambridge MA, 1989.
8. D.P. Darbha, S.; Agrawal. Optimal scheduling algorithm for distributed-memory machines. *IEEE Transactions on Parallel and Distributed Systems*, 9(1):87 – 95, January 1998.
9. Jing-Chiou Liou and M.A. Palis. A new heuristic for scheduling parallel programs on multiprocessor. In *1998 International Conference on Parallel Architectures and Compilation Techniques*, pages 358 – 365, October 1998.
10. Virginia M. Lo. Temporal Communication Graphs: Lamport's process-time graphs augmented for the purpose of mapping and scheduling. *Journal of Parallel and Distributed Computing*, 16(4), December 1992.
11. Tao Yang and Cong Fu. Heuristic algorithms for scheduling iterative task computations on distributed memory machines. *IEEE Transactions on Parallel and Distributed Systems*, 8(6):608–622, June 1997.
12. A. Gerasoulis and T. Yang. A comparison of clustering heuristics for scheduling DAGs on muliprocessors. *Journal of Parallel and Distributed Computing*, 16(4):276–291, December 1992.
13. A. A. Khan, Carolyn L. McCreary, and M. S. Jones. A comparison of multiprocessor scheduling heursitics. Technical Report ncstrl.auburn_eng/CSE94-02, Department of Compiter Science and Engineering, Auburn University, Auburn, AL 36849, January 1994.
14. Y. Kwok and I. Ahmad. Benchmarking the task graph scheduling algorithms. In *Proceedings of the 1st Merged Int. Parallel Processing Symposium and Symposium on Parallel and Distributed Processing (IPPS/SPDP-98)*, pages 531–537, Orlando, Florida, USA, April 1998.
15. Yu-Kwong Kwok and Ishfaq Ahmad. A comparison of parallel search-based algorithms for multiprocessors scheduling. In *Proceedings of the Second European Conference on Parallel and Distributed Systems (EURO-PDS'98)*, Vienna, Austria, July 1998.
16. Gilbert C. Sih and Edward A. Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Transactions on Parallel and Distributed Systems*, 4(2):175–186, February 1993.
17. Yu-Kwong Kwok and Ishfaq Ahmad. Bubble scheduling: A quasi dynamic algorithm for static allocation of tasks to parallel architectures. In *Proceedings of the 7th IEEE Symposium on Parallel and Distributed Processing (SPDP'95)*, pages 36–43, Dallas, Texas, USA, October 1995.
18. Oliver Sinnen and Leonel Sousa. A comparative analysis of graph models to develop parallelising tools. In *8th IASTED Int'l Conference on Applied Informatics (AI'2000)*, Innsbruck, Austria, February 2000.
19. Oliver Sinnen and Leonel Sousa. A classification of graph theoretic models for parallel computing. Technical Report RT/005/99, INESC, Instituto Superior Técnico, Technical University of Lisbon, Lisbon, Portugal, May 1999.

20. Sun Yuan Kung. *VLSI Array Processors.* Information and System Sciences Series. Prentice Hall. Englewood Cliffs, New Jersey 07632, 1988.
21. Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565. July 1978.
22. Virginia M. Lo, Sanjay Rajopadhye, Samik Gupta, David Keldsen, Moataz A. Mohamed, Bill Nitzberg, Jan Arne Telle, and Xiaoxiong Zhong. OREGAMI: Tools for mapping parallel computations to parallel architectures. *International Journal of Parallel Programming*, 20(3):237–270, June 1991.
23. Utpal Banerjee, Rudolf Eigenmann, Alexandru Nicolau, and David A. Padua. Automatic program parallelization. *Proceedings of the IEEE*, 81(2):211–243, February 1993.
24. Yu-Kwong Kwok and Ishfaq Ahmad. Efficient scheduling of arbitrary task graphs to multiprocessors using a parallel genetic algorithm. *Journal of Parallel and Distributed Computing*, 47(1):58–77, November 1997.
25. Lee Wang, Howard Jay Siegel, Vwani P. Roychowdhury, and Anthony A. Maciejewski. Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach. *Journal of Parallel and Distributed Computing*, 47(1):8–22, November 1997.
26. Keshab K. Parhi. Algorithm transformation techniques for concurrent processors. *Proceedings of the IEEE*, 77(12):1879–1895, December 1989.
27. Keshab K. Parhi and David G. Messerschmitt. Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding. *IEEE Transactions on Computers*, 40(2):178–195, December 1991.

A Tool for Distributed Software Design in the CORBA Environment

Jan Kwiatkowski[1,2], Maciej Przewoźny[2], Jose C. Cunha[3]

[1] Institute of Computer Science
University of Wrocław
51-151 Wrocław, Przesmyckiego 20, Poland
[2] Computer Science Department
Wrocław University of Technology
50-370 Wrocław, Wybrzeże Wyspiańskiego 27, Poland
tel. (+48)(713203602, fax: (+48)(71)3211018
Emails: {przewozny, kwiatkowski}@ci.pwr.wroc.pl.
[3] Computer Science Department
University Nova of Lisbon
2825 Monte da Caparica, Portugal
tel: 351 (1) 2943220, fax: 351 (1) 2948541
E-mail: jcc@di.fct.unl.pt

Abstract. Nowadays Distributed Object Oriented Environments are becoming widely used. One of them is OMG's Common Object Request Broker Architecture (CORBA). The paper deals with the short description of a tool named U_CORBA, which supports the design process of CORBA applications. The new tool was built under the MICO CORBA implementation. It gives opportunity of creation of the class diagram of the application using the UML notation and based on it generating the IDL files and application C++ headers. The tool also includes some management functions, which allow a user an easy way to visualize the state of the CORBA environment as well as managing it.

1 Introduction

Nowadays Object Oriented models are being used in many applications. On the other hand with the increased necessity of enabling computers to work together a new kind of programming and working environments are now becoming widely used - the Distributed Object Oriented Environments. It is still too early to make guesses on which one of these systems will become the standard. So far the strongest contestants are IBM's System Object Model (SOM); Microsoft's Distributed Object Linking and Embedding (OLE); OMG's Common Object Request Broker Architecture (CORBA). Each one of them has their advantages and drawbacks. The paper deals with the short description of a tool named U_CORBA, which supports the design of CORBA applications. The CORBA environment was chosen, as it is the most advanced one in terms of standard definition. Unlike some others, it has already available full programming and runtime environments.

CORBA brought new concept used for the creation of applications in a distributed manner. It is a solution for developing new application or making together some working application [5,8]. Although there are a lot of different developing methodologies and tools supporting application design, like UML, VPE, TRAPPER and others working at different environments there are no such tools dedicated for developing CORBA applications [2,3,4,7,9]. For designing a CORBA application, Rational Rose is a commonly used environment [7]. This is the main motivation of the work presented in the paper. The main objective of this work is to design and implement a graphical tool that helps in developing CORBA applications. The presented tool named U_CORBA gives opportunity to create the class diagram of the application using the UML notation and based on it generating the IDL files and application C++ headers. Additionally the tool includes some management functions, which allow a user an easy visualization of the state of the CORBA environment as well as of managing it. The accessible functions are similar to these ones which are at OrbixManager from IONA Technologies [1]. It gives a user some benefits in designing the CORBA applications during the development process. The prototype of the tool was developed with the QT 1.44 library and MICO 2.2.5 [5] on Linux 2.2.5. The structure of this paper is as follows. In section 2, a brief overview is presented of the CORBA architecture. In section 3, the main functionalities of the U_CORBA tool are described. Some implementation details are described in section 4. Section 5 presents an example of the use of the tool. Section 6 compares the tool to related tools supporting Object Oriented development. Finally, section 7 presents some conclusions and describes ongoing work.

2 CORBA Overview

CORBA stands for Common Object Request Broker Architecture and is a platform defined by OMG - the Object Management Group, a consortium of several companies and universities working together in its definition. The main purpose of the OMG is to define a platform for heterogeneous distributed computing in which very different hardware will work smoothly together - from super computers to embedded systems - independently of the operating systems, programming languages and network protocols they might be using.

In CORBA objects interact with each other by means of *interface* definitions, with the information provided by these interfaces, potential clients of object services are able to know what to expect from objects and how they should interact with them. This interface is defined in OMG's IDL (Interface Definition Language) which enables object services to be available to other objects written in almost any programming language. By using IDL, the programmer lets the communication infrastructure know the format of all messages an object can receive and send so that, if necessary, they can automatically be transformed from one data representation to another, providing transparent communication between different systems. The communication infrastructure defined by OMG is called the OMA (Object Management Architecture) and it is a set of protocols and services definitions that allow very different objects to interact freely with each other. OMG defined a set of

standard interfaces and functions for each component of the OMA. All CORBA services communicate with each other through an *ORB* (Object Request Broker) that handles and delivers all messages from one component to another so programmers do not have to worry about distribution details, and can concentrate on solving the real problem at hand.

When designing a CORBA application two files are generated: a skeleton and a stub (figure 1). These files, when compiled and linked together with the service implementation will act as translators between the object and the ORB. Because of this the clients and object implementation can even be written in different programming languages, one just has to generate the skeleton and stubs using the appropriate IDL compilers.

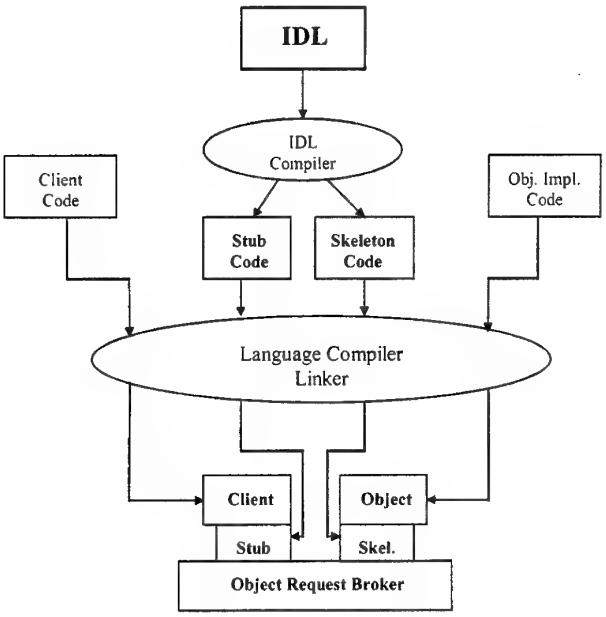


Fig. 1. The structure of CORBA application

There are currently some *official* IDL languages mapping specifications standardised by OMG: for Java, C, C++, Smalltalk, Ada. Besides these there are other mappings currently not supported by any standard, and many other IDL compilers are specific for a given ORB. As OMG does not force any communication protocol between the skeletons and the ORB (the IIOP protocol is in common use), a given skeleton will only be able to *talk* with its corresponding ORB. What happens if the ORB being used is changed? That is not a problem, new skeletons must be generated using the new IDL compiler and linked with unchanged object implementations - thus achieving instant integration. Both client and object implementation are isolated from

the ORB by IDL interfaces so that the client does not even have to care about the way objects are implemented, making modifications easy.

For our purpose we have chosen the MICO CORBA implementation, which is a complete CORBA 2.2 compliant implementation. MICO is freely available. The difference to other freely available implementations is that MICO is developed for educational purposes and that the complete sources are available under the GNU-copyright notice. Among free ORB with C++ mapping this one provides the most impressive list of features:

- Modular ORB design: new transport protocols and object adapters can easily be attached to the ORB – even at runtime using loadable modules,
- It offers an interface for inserting and extracting constructed types that were not known at compile time,
- Full BOA implementation, including all activities modes, support for object migration, object persistence and the implementation repository,
- BOA can load object implementation into clients at runtime using loadable modules.

3 Application Design Using U_CORBA

The presented tool helps CORBA users in developing CORBA applications (servers and clients) by generating the skeleton of the application by means of IDL files and application C++ headers. A specially designed graphical interface gives the user opportunity of creating the application class diagrams using UML notation and based on it generating the IDL and application C++ header files. The tool is equipped with an embedded user's editor for writing application body. The main window of the application containing three parts is presented in figure 2. On the top there is a popup menu. First part of the menu gives the basic editor functions like creation of the new

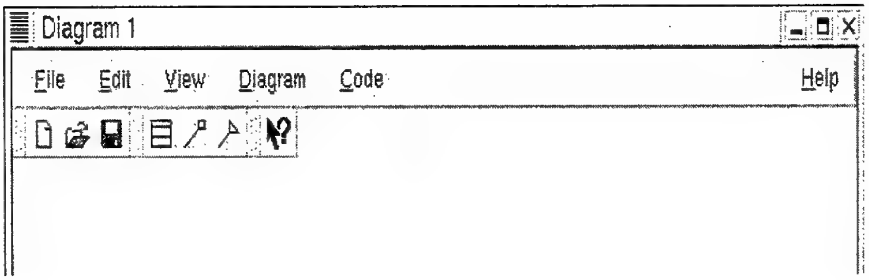


Fig. 2. The main window of U_CORBA tool

diagram, opening of the saved diagram, saving prepared diagram, printing the diagram and quitting the tool applications. Options for opening, saving and printing open the standard dialogs for choosing the files and printers. Then there is the group of options for editing the documents containing such common used functions like: *Cut, Copy, Paste*, etc. The *View* option allows modifying the appearance of the tool

on the screen. The specific functions of the tool are accessible using *Diagram* and *Code* options. Using the *Diagram* option the user can create a class diagram and when using the *Code* option can generate IDL files and C++ application headers and writes the body of an application. Every option starts the specific action for the chosen functionality to be performed. For example *New Class* from *Diagram* option opens the dialog in which the information about the new class is collected.

The toolbar contains iconic shortcuts for a couple of the menu options. Starting from the left side there are:

- New Diagram – creates the new application diagram,
- Open – starts the open file chosen dialog
- Save – saves editing diagram,
- New Class – starts the new class dialog,
- New aggregation – inserts the new aggregation,
- New Generalization – inserts the new generalization,
- Help – gets the information about accessible options.

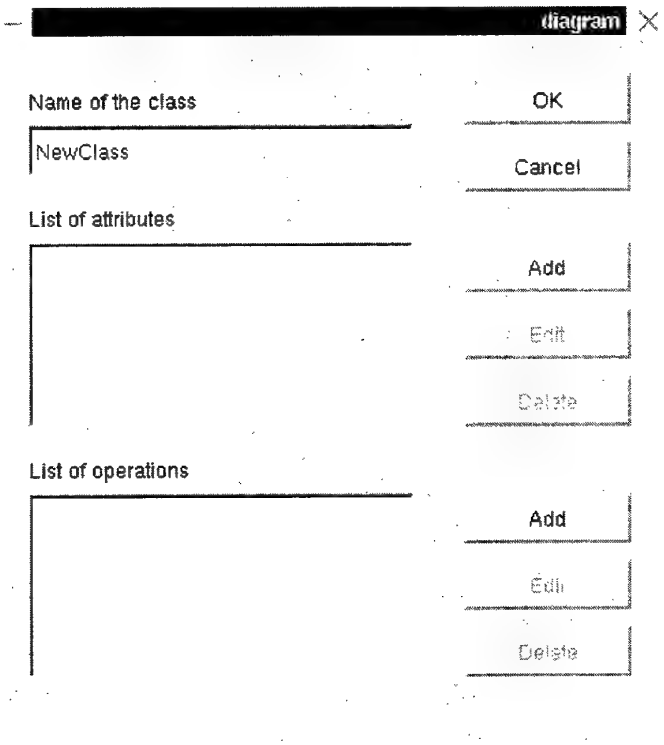


Fig. 3. The *New Class* dialog window

Next part of the main window in the central widget where the diagram is draw. The appropriate way of drawing the elements of the diagram is the standard described by

the OMG. When the new class is inserted to the diagram the set of dialogs is used to collect information about class properties (figure 3).

This dialog contains the line editor for the name of the class specifying two list boxes that contain all the attributes and operations. The attributes and operations are performed by the set of buttons (*Add, Edit, Delete*). All actions performed in the dialog can be confirmed by the *OK* or discarded by *Cancel* buttons, respectively. After adding or editing the list of attributes or operations, a new dialog box is opened for attributes or operations specification. In the attribute dialog box (figure 4) the information about attribute properties is collected. The combo boxes provide the set of choices for the attribute type and export control. The default type is set to "int" and export control to "private". A similar dialog box is defined for operations. Theirs functionality is similar to the one described in the *Attribute* dialog. The default operation type is set to "int" and export control to "public".

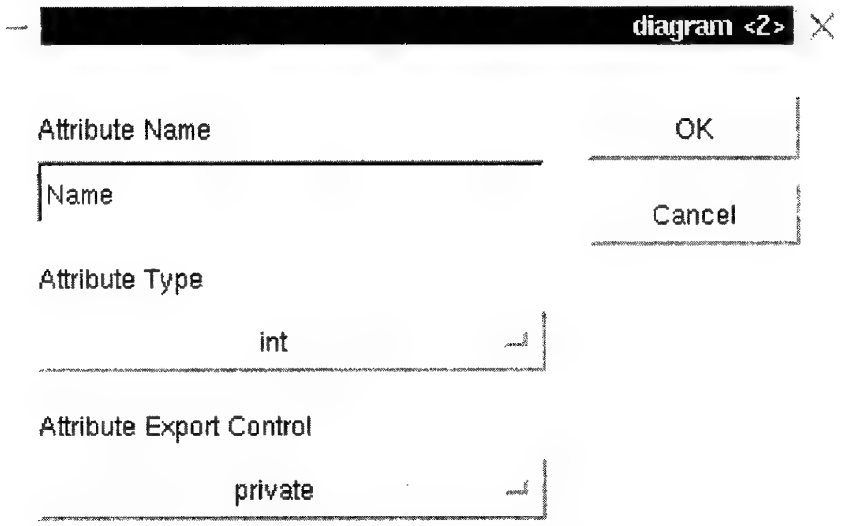


Fig. 4. The *Attribute* dialog window

In contrast to other available in CASE tools code generators our CORBA IDL generator (*Code* option) produces only CORBA IDL specification and C++ headers files. To produce executable code after translation of IDL files to architecture specific C++ headers and building the implementation using programmer editor the source code of the application is ready to compilation.

Presently the tool covers only a part of the functionalities that are normally supported by CASE tools, however experiments performed using our prototype indicate that it is useful during the design of CORBA applications. The prototype is being extended to support all functionalities required by a full development process.

4 Some Implementation Details

Presented in the paper tool consist of two parts, the management and development tools. The management tool allows a user an easy visualization of the state of the CORBA environment as well as of managing it and the development tool supports the process of design the CORBA application by drawing the class diagram and generating the IDL files and application C++ headers. Figure 5 gives an overview of the U_CORBA environment layers, arrows indicate the communication between different environment components. Tools do not directly call the system or MICO services. The manager tool operates on the MICOs Implementation Repository (IR) through the program provided in the MICO package, which is called "imr" and calls the routines from the QT libraries for graphics and from "imr" for operations on the IR. The class diagram that represents the tool functionalities cooperates only with the QT library since it provides all the required functionality. The main tasks of the manager are:

- The management of the implementation repository daemon,
- The management of entries into the implementation repository,
- Visualization of the current status of implementation repository daemon.

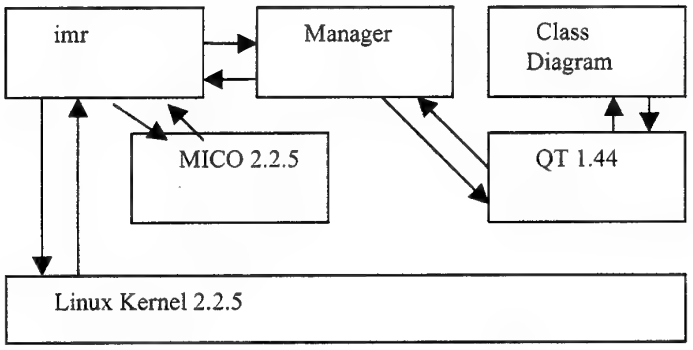


Fig. 5. Layers of U_TOOL

There are two main problems that appear during developing tool implementation: definition of the data structures for class diagram representation and the translation grammar for the application C++ headers and IDL code generation.

The attribute name and type describe the class attributes. The name is stored in the string and the type can be set to one of many possible options. That set includes the most popular types like *int, float, string, char*, etc. The class abstracting the attribute has to be equipped with the copy constructor and overloaded assignment operator for the handling of these structures is not just the question of bit to bit copy. Such solution gives the opportunity of future tool development. Considering attributes in context of the OOP one more feature has to be added - a visibility. Visibility specifies the way of accessing the attributes. The name, type, and list of attributes and visibility describe the operations. Name, type and visibility describing operations are stored in similar way as for attributes, but the list of attributes causes more problems. It can be

empty or the number of eventual attributes is unknown. Then operation attributes are stored in the list. The list contains only the pointers to the objects. That requires a lot of caution while processing the entities of operations. The list itself is hold by the type provided by the Qt toolkit.

The code is generated for the specific class on the class diagram. The name of that class indicates the name of the file. Different suffixes are added depending on the file type generating *.*idl* for the IDL files and *.*h* for the application C++ headers. The process of file generation is divided into three steps. In the first step the class name using the above described procedure is established. During the second step information about the links between classes is retrieved from the association database created from the class diagram, it causes some extra lines of code at both generated files. In the last step the analysis of attributes and operations is performed. As long as the number of used types for the attributes or operations is limited to the basic ones the process of generation of both files is not complicated. However the generation of application C++ headers for complex types is not so obvious, usually attributes and operations belonging to the one type of visibility are grouped in one place, and depending on the convention those groups are placed in different places in the generated header file. We assume that for the more clear declaration reading the group with *public* visibility comes first.

5 An Example of Using U_CORBA

To present the functionalities provided by a tool, we present the following two simple examples. Let's consider the simplified version of a bank account server. It is responsible for managing a user account by depositing and withdrawing the required sum of money, and reporting about the current state of the account only.

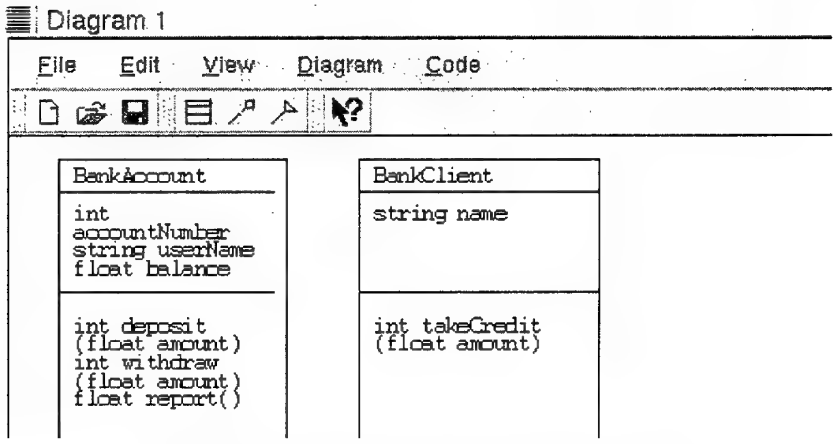


Fig. 6. Class diagrams for the server and client respectively

The server needs to hold some information, like current balance, account identification number, etc. Let's assume that a bank client is interested in obtaining the credit. Then both of them can be represented as objects in CORBA environment as presented in figure 6.

After defining both class diagrams using the implemented graphical environment using *Code* option the IDL files and C++ application headers for server and client are generated.

The IDL files generated for the server and client respectively are presented below.

```
  interface bankAccount
{
  attribute long accountNumber;
  attribute string userName;
  attribute float balance;
  long deposit(in float amount);
  long withdraw(in float amount);
  float report();
}
interface bankClient
{
  attribute string name;
  long takeCredit(in float amount);
}
```

The application C++ headers generated for the server and client respectively

```
class bankAccount
   {
    private:
      long accountNumber;
      string userName;
      float balance;
    public:
      long deposit(float amount);
      long withdraw(float amount);
      float report();
   }
class bankClient
   {
    private:
      string name;
    public:
      long takeCredit(float amount);
   }
```

After translation of generated IDL files to the architecture specific C++ headers the user can start to build the implementation of the server and client behavior using the *Code* option of the tool (programmer editor).

As a second example let's consider the management system of relational database (RDBMS). The class diagram for an example is presented at figure 7. The management system consists of some servers (DB) distributed among the different computers (multiserver). Used at the diagram the diamond symbol represents aggregation, when a triangle symbol represents inheritance.

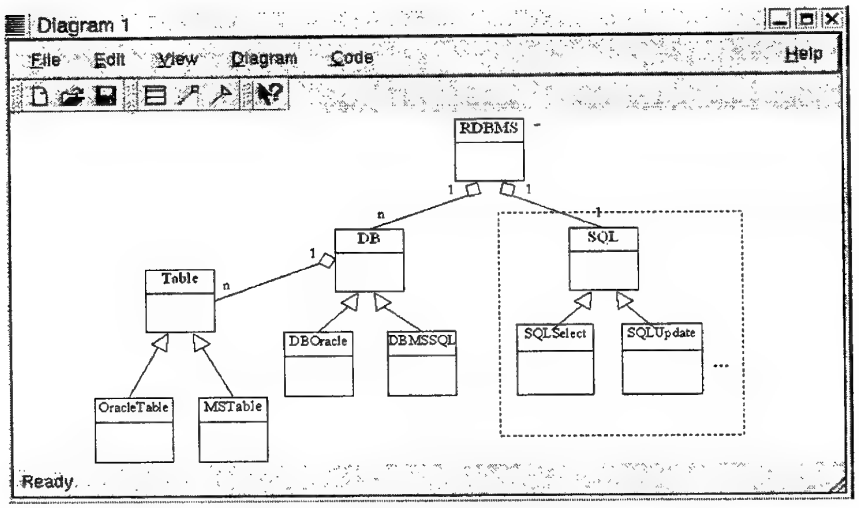


Fig. 7. Class diagram for DBDMS system

The way of data partitioning between different servers can be vertical or horizontal and data can be additionally replicated. Transactions in such distributed environment are of two types: *queries*, which involve request for information and *updates*, which generate changes to the data entries in the database. The queries can be proceeded locally or it may be necessary to access remote sites. Then the "SQL module" represents the CORBA client. Different CORBA objects (databases, tables) can be moved in the whole distributed environment (middleware interface).

The part of generated by the tool IDL file for the above described database management system is presented below.

```
 module RDBMS
{
 // abstract table
 interface Table
 {
 attribute string strName;     // name of the table
 attribute boolean Temp;       // flag for temp. sector
 void addRow(in long lSpace); // space for the new row
 };
.........
 // abstract database
 interface DB
 {
 attribute string strName;      // name of the DB
 attribute Table intfcTable[]; // set of Tables
 void ClearTempSector();        // operation that
        cleans  the temporary sector of the database
 void AllocateSpace(in Table forTable);  // allocates
                                physical space
};
```

```
 interface DBMSSQL : DB
{
 attribute boolean bAllowTruncate;     // flag to allow
            truncation of the objects (like table)
 void setTruncate(in boolean yesno);    // allow or
                                        disallow
};
.........
};
```

6 Comparing U_CORBA with Other Tools

The main aims of using development-supporting tools can be pointed by:

- Accelerating the development time, by improving communication among various team members,
- Improving quality, by mapping business processes to software architecture,
- Increasing visibility and predictability by making critical design decisions explicit visually.

Comparing our tool with other available tool we can conclude that its functionalities with respect to supporting the CORBA application development process is comparable. In Rational Rose some functions supporting the CORBA 2.2 application development are included. It supports forwarded and reverse engineering of CORBA IDL. It supports CORBA specific function such as; Stereotypes, Constants, Enums, Exceptions, Interfaces, Structs, Typedefs and Unions. It includes a built-in color-coding editor, which allows editing of IDL syntax files from within Rose 98 [7]. Similar functionalities can be found in COOL-Jex, which support developing of CORBA application by generating IDL files for forwarded as well as for reverse engineering [9]. From the other hand the whole functionalities such tool like Rational Rose is much wider and include component-based development, multi-language development, UML modeling, etc. The main advantage of using our tool is that it supports not only developing process but also some useful CORBA environment management functions are provided.

7 Conclusions

This is an ongoing project. Presently we finished building a prototype. The prototype still misses a lot of features that could be implemented in the further versions. However, experiments performed using our prototype indicate that the presented tool will be useful for designing CORBA applications. No specific knowledge about the IDL language is required to build CORBA objects. After translation of the IDL files to the architecture specific C++ headers the user can start to build the implementation of the server/client behavior. The tool helps the developer out in IDL knowledge and allows him to concentrate on the essential part of implementation. The decision of using OO Technology for coding and a GPL

implementation of the CORBA system will contribute to ease the implementation of the full set of desired functionality. Moreover, those decisions make the development of further aspects and kinds of experiments possible, since all layers of the implementation are accessible. Having the Rose tool so sophisticated and supporting so many features and recently also ported to the Unix systems, the question about the motivation of building other tool with a similar functionality arises. With a tool released under the General Public Licence it is possible to freely access the code, develop new features and made it more efficient. The dedication to the MICO that we have followed in this project allows taking some benefits and widening the functionality to areas that are not possible for a tool that is so general like Rose. That all opens a chance for a flexible support for developers that are not capable of obtaining the commercial expensive tools.

References

1. Baker S., "Distributed Objects using Orbix", Addison-Wesley, 1997
2. Erikson H., Penker M., „UML Toolkit", John Wiley & Sons, Inc., 1998
3. Douglass B., „Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns", Addison-Wesley, Inc. 1999
4. Newton P, Dongarra J., „Overviev of VPE: A Visual Environment for Message-Passing Parallel Programming", http://www.netlib.org/tennesse/ut-cs-94-261.ps
5. OMG, „The Common Object Request Broker: Architecture and Specification - OMG IDL Syntax and Semantics" 1998
6. Puder A,. „MICO Online Documentation", 1999
7. Rational Software Corporation - Rational Rose 98 Help
8. Siegel J., „CORBA Fundamentals and Programming", John Wiley & Sons, Inc., 1996
9. Sterling Software Corporation – COOL :Jex Help
10. Tanenbaum A., „Distributed Operating Systems", Prentice Hall, 1995
11. Zomaya A., Parallel and Distributed Computing Handbook, McGraw-Hill, 1996

Parallel Performance of Ensemble Self-Generating Neural Networks

Hirotaka Inoue and Hiroyuki Narihisa

Department of Information & Computer Engineering, Faculty of Engineering,
Okayama University of Science,
1-1 Ridai-cho, Okayama 700-0005, Japan
{inoue, narihisa}@ice.ous.ac.jp

Abstract. In this paper, we investigate the improving capability of accuracies and the parallel efficiency of ensemble self-generating neural networks (ESGNNs) for classification problems and the time series prediction on a MIMD parallel computer. The results of our computational experiments show that the more the number of processors increases, the more the improvement of the accuracy is obtained for all problems, and the parallel efficiency is obtained for all problems.[1]

1 Introduction

Neural networks have been widely used in the field of the intelligent information processing such as classification, clustering, prediction, and recognition. Generally, neural networks have to be decided the network structures and some parameters by human experts. It is quite tricky to choose the right structure of neural networks suitable for a particular application at hand. In order to avoid these tricky and difficult situations, self-generating neural networks (SGNNs) are focussed an attention because of their simplicity on networks design [1]. SGNNs are some kinds of extensions of the self-organizing maps (SOMs) of Kohonen [2] and utilize the competitive learning algorithm which is implemented as a self-generating neural tree (SGNT).

The SGNT algorithm is proposed in [3] to generate a neural tree automatically from training data directly. Originally, this SGNT algorithm is developed as a hierarchical clustering algorithm. Therefore, it may be a natural consequence to show a good performance in applying to the classification or clustering problems. In our previous study concerning the performance analysis of the SGNT algorithm [4], we showed that the main characteristic of this SGNT algorithm was its high speed convergence in computation time but it was always not best algorithm in its accuracy comparing with the existing other feed-forward neural networks such as the backpropagation (BP) [5]. In order to acquire more higher accuracy of SGNNs, we introduced the ensemble averaging approach to

[1] Candidate to the Best Student Paper Award

improve the generalization capability of SGNNs which is fully utilize the high speed convergence characteristic of the SGNT algorithm [6].

In this paper, we investigate the improving capability of accuracies and the parallel efficiency of ensemble self-generating neural networks (ESGNNs) for classification problems and the time series prediction on a MIMD parallel computer. We analyze MONK's [7] problems in classification and the Mackey-Glass time series [8] in time series prediction which are given as benchmarks.

This paper is organized as follows: Section 2 outlines the learning system on neural networks and describes accuracies as criteria for evaluation for classification and time series prediction. Section 3 describes SGNNs. In Section4, we combine the ensemble averaging method with the SGNN model in order to improve the generalization capability. In Section 5, we present how to perform the ESGNN on the parallel computer. Section 6, we describe experimental details. Section 7 is devoted to investigate the improving accuracy and parallel performance for ESGNNs through a simulation study, and Section 8 concludes the paper with some remarks.

2 Learning System and Accuracies

A training data set D consists of data $\{(\boldsymbol{x}_i, y_i), i = 1, \ldots, N\}$ and a test data set T consists of data $\{(\boldsymbol{x}_i, y_i), i = 1, \ldots, M\}$. Here, $\boldsymbol{x}_i$ is the input and y_i is the desired output, and factors of D and T are independent of each other. The learning task is to construct a learning system from this training data set D in order to classify/predict y by the output of this learning system $f(\boldsymbol{x})$ for system input $\boldsymbol{x}$. After constructing the learning system, then the accuracy of this learning system is evaluated by the test data set T (see Fig. 1). Next, we show how to evaluate the performance of the learning system for classification and time series prediction, respectively.

In classification, the objective of the learning system is to classify successfully on the test data set T. The input data $\boldsymbol{x}_i$ corresponds to the discrete p-dimensional attributes vector and the output y_i corresponds to a class label, for $(\boldsymbol{x}_i, y_i) \in T$. If $f(\boldsymbol{x}_i)$ and y_i are the same label, this case may be considered to be success, otherwise failure. Therefore, the accuracy of this learning system is evaluated by counting these success and failure cases for a given test data. The most commonly used criterion for the accuracy of classification system is misclassification rate which is the ratio of the number of failures to the number

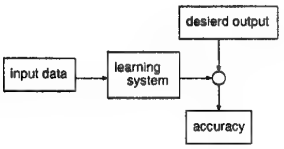


Fig. 1. Functional diagram of the learning system

of test data as follows:

$$\text{misclassification rate} = \frac{\text{number of failures}}{\text{number of test data}} . \tag{1}$$

In this paper, we use above defined misclassification rate.

In time series prediction, the objective of the learning system is to predict the future on the test time series data set T. The input data $\boldsymbol{x}_i$ involves processing of patterns that evolve over time. In neural networks prediction, temporal information of the series data is spatially brought to the network by a p-dimensional time-lagged vector;

$$\hat{x}_{t+L} = f(x_t, x_{t-m}, \ldots, x_{t-(p-1)m}) , \tag{2}$$

where $\hat{x}_{t+L}$ is the approximation of the real time series data x_{t+L} on time $t+L$. Here, x_{t+L} is corresponding to the desired output y_i for $\boldsymbol{x}_i$. As a criterion for time series prediction system, we use the following ARV (average relative variance) which is commonly used in this time series prediction community [9],

$$\text{ARV} = \frac{\sum_{i \in T}(y_i - f(\boldsymbol{x}_i))^2}{\sum_{i \in T}(y_i - \text{E}[y_i])^2} , \tag{3}$$

which is the mean squared error (MSE) divided by the variance of desired outputs on the test data set T. Here, E is the expectation value on statistical sense, y_i is the real time series data and $f(\boldsymbol{x}_i)$ is the output of the learning system, respectively.

3 Self-Generating Neural Networks

SGNNs proposed in [3] are based on SOMs and traditional AI unsupervised learning methods such as COBWEB [10]. SGNNs are implemented as a self-generating neural tree (SGNT) architecture. Generally, the SGNT algorithm has no learning parameters. The structure of the SGNT changes dynamically in training. The SGNT algorithm decide the structure of the SGNT after all training data are added in leaves of the SGNT.

The SGNT algorithm is defined as a tree construction problem how to construct a tree structure from the given data which consist of multiple attributes under the condition that final leaf neurons correspond to the given data. Before we describe the SGNT algorithm, we denote some notations.

- input data vector : $\boldsymbol{e}_i$; $\boldsymbol{e}_i = (e_{i1}, e_{i2}, \ldots, e_{ip})$.
- j-th neuron : n_j; n_j is expressed as ordered pair $(\boldsymbol{w}_j, c_j)$.
- weight vector of n_j : $\boldsymbol{w}_j$; $\boldsymbol{w}_j = (w_{j1}, w_{j2}, \ldots, w_{jp})$.
- the number of the leaf neurons in n_j : c_j.
- tree is expressed as ordered pair $(\{n_j\}, \{l_k\})$, where $\{n_j\}$ is the neuron set and $\{l_k\}$ is the link set of the tree.
- distance measure : $d(\boldsymbol{e}_i, \boldsymbol{w}_j)$; we use Euclidean distance measure.

– winner neuron for e_i : n_{win}.

The SGNT algorithm is a hierarchical clustering algorithm. The pseudo C code of the SGNT algorithm is given as follows:

Algorithm (SGNT Generation)

```
Input :
  A set of training examples E = {e_i}, i = 1, ... , N.
  A threshold value XI >= 0.
  A distance measure d(e_i,w_j).
Program Code :
  copy(n_1,e_1);
  for (i = 2, j = 2; i <= N; i++) {
    n_win = choose(e_i, n_1);
    minDistance = distance(e_i, w_win);
    if (minDistance > XI) {
      if (leaf(n_win)) {
        copy(n_j, w_win);
        connect(n_j, n_win);
        j++;
      }
      copy(n_j, e_i);
      connect(n_j, n_win);
      j++;
    }
    update(e_i, w_win)
  }
Output :
 Constructed SGNT by E
```

In the above algorithm, some sub procedures are used. Table 1 shows the sub procedures of the SGNT algorithm and their specifications.

In order to decide the winner neuron n_{win}, competitive learning is used. If a n_j includes the n_{win} as it's descendant in the SGNT, the weight w_{jk} ($k =$

Table 1. Sub procedures of the SGNT algorithm

Sub procedure	Specification
$copy(n_j, \boldsymbol{e}_i/\boldsymbol{w}_{win})$	Create n_j, copy attributes of $\boldsymbol{e}_i/\boldsymbol{w}_{win}$ as weights $\boldsymbol{w}_j$ in n_j.
$distance(\boldsymbol{e}_i, \boldsymbol{w}_j)$	Compute $d(\boldsymbol{e}_i, \boldsymbol{w}_j)$.
$choose(\boldsymbol{e}_i, n_1)$	Decide n_{win} for $\boldsymbol{e}_i$.
$leaf(n_{win})$	Check n_{win} whether n_{win} is leaf or not.
$connect(n_j, n_{win})$	Connect n_j as child neuron of n_{win}.
$update(\boldsymbol{e}_i, \boldsymbol{w}_j)$	Update $\boldsymbol{w}_j$ of n_j.

$1, 2, \ldots, p$) of the neuron n_j is updated as follows:

$$w_{jk} = w_{jk} + \frac{1}{c_j + 1} \cdot (e_{ik} - w_{jk}). \tag{4}$$

After all training data are inserted into the SGNT as leaf neurons, the weights of each node neuron n_j is the averages of the corresponding weights of all its children. The whole network of the SGNT reflects the given feature space by it's topology.

In the SGNT, the input data $\boldsymbol{x}_i$ corresponds to $\boldsymbol{e}_i$, and the desired output y_i corresponds to the network output o_i which is stored in one of the leaf neuron, for $(\boldsymbol{x}_i, y_i) \in D$.

In the testing process, the input data $\boldsymbol{x}_i$ is entered the root neuron of the SGNT as $\boldsymbol{e}_i$. Then the input data are reached one of the winner leaf neuron n_{win} of the SGNT through competition, and the desired output y_i is compared with the network output o_{win} in order to evaluate the accuracy of the SGNT. , for $(\boldsymbol{x}_i, y_i) \in T$. Note that though the competitive learning of the training process is performed among a parent and it's children recursively, the competitive learning of the testing process is performed among only children recursively.

4 Ensemble Averaging

SGNNs have some abilities as follows:

- fast learning,
- learning stability,
- good mapping of given input data in the tree structure.

However, because of SGNT algorithm is originally based on an unsupervised learning method, the accuracy of the classification/prediction is not so good as feed-forward neural networks which are implemented as a supervised learning method like BP.

In order to acquire more higher generalization ability, we adopt ensemble averaging method [11] to SGNNs. The ensemble averaging method is based on statistic theory. This method proofed as following theories in [12]:

1. The bias of the ensemble averaged output, pertaining to the ensemble, is exactly the same as that of the output pertaining to a single neural network.
2. The variance of the ensemble averaged output is less than that of all single neural network.

Here, the bias and the variance are decomposition components of MSE as follows [13]:

$$\text{MSE} = B + V; \tag{5}$$

$$B = (\mathrm{E}_D[f(\boldsymbol{x})] - \mathrm{E}[y|\boldsymbol{x}])^2, \tag{6}$$

$$V = \mathrm{E}_D[f(\boldsymbol{x}) - \mathrm{E}_D[f(\boldsymbol{x})]^2], \tag{7}$$

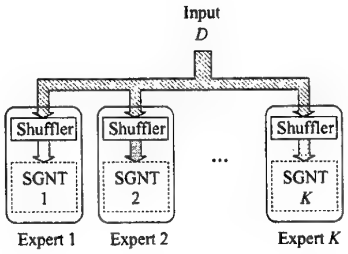


Fig. 2. Ensemble of K SGNTs (training process). One expert corresponds to one SGNT, the shuffler makes shuffle elements of input data

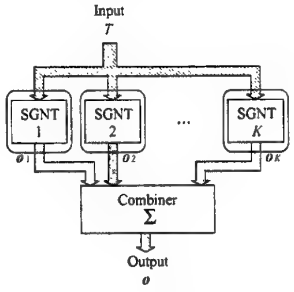


Fig. 3. Structure of the ensemble system (testing process)

where E is the expectation value on statistical sense , B is the bias decomposition of MSE, and V is the variance decomposition of MSE. These theories means that the overall error produced by an ensemble model are improved by averaging the output of all single network in the ensemble model.

Next, We describe how to make the ensemble SGNNs model. This model can separate a training process and a testing process. In the training process, we define "shuffler" to shuffle the set of input data D. Fig. 2 shows the structure of the ensemble system including K SGNTs on a training process. The set of all input training data D enters each SGNN through each shuffler. The shuffler makes shuffle elements of D at random. All SGNTs are generated by adopting the SGNT algorithm. After the training process, various SGNTs are generated independently.

In the testing process, the set of test data T is entered this ensemble model. Fig. 3 shows the structure of the ensemble system including K SGNTs on the testing process. Each output vector $\boldsymbol{o}_k \in \Re^M$ denotes the output of the expert k for the set of test data T. Here, M denotes the number of test data. The output of this ensemble model is computed by averaging the each expert output as follows:

$$\boldsymbol{o} = \frac{1}{K}\sum_{k=1}^{K} \boldsymbol{o}_k. \tag{8}$$

In this paper, we adopt the ensemble model to three binary classification problems and the time series prediction. Considering the classification problems, in order to classify each test data, corresponding output $o_i (i = 1, \ldots, M)$ is evaluated as follows:

$$\begin{aligned} o_i \geq 0.5 \quad & : \text{Class1}, \\ o_i < 0.5 \quad & : \text{Class0}. \end{aligned}$$

5 Parallelization of ESGNNs

Because of each expert of ESGNNs can train and test independently, the ESGNNs model has a possibility of the parallel computation at the training process and the testing process. Hence, we allocate each of experts to each of processors on the MIMD computer. The procedure of the parallelization of ESGNNs is presented as follows:

Step1: In a master processor, read the training set D and the test set T in the disk.
Step2: In the master processor, broadcast D and T for all $K-1$ slave processors.
Step3: In all processors, generate the SGNT from D, then test the SGNT using T, and compute the $\boldsymbol{o}_k$ independently.
Step4: In all processors, each output $\boldsymbol{o}_k$ for T is collected in the master processor by all to one communication.
Step5: In the master processor, compute $\boldsymbol{o}$ by Eq. (8) and write to the disk.

Because of the number of the communications between the master processor and each slave processor is only two times (Step2 and Step4), the parallel efficiency is approximately expected the linear speedup. (See Fig. 4) In our case, all computations are performed on the Intel Paragon (Paragon XP/S15). This is a distributed memory multicomputer, and the architecture is multiple instruction multiple data (MIMD). The Paragon we use has 296 processors. Each processor is Intel i860XP (50MHz). The network topology of the Paragon is adopted the two-dimensional mesh.

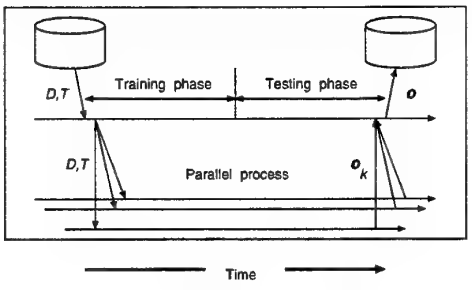


Fig. 4. Parallelization of ESGNNs

6 Experimental Details

We allocate a SGNT to each of processors on the Paragon, and compute 100 trials for each single/ensemble model. The number of processors (SGNTs) K for the ensemble averaging is changed from 1 to 30 (1,2,3,4,5,6,7,8,9,10,15,20,25, and 30), and the threshold value ξ is 0 for each SGNT algorithm. Because of the redundancy reduction, we repeated 100 trials from Step3 to Step5 in prior section continuously. Generally, the parallel efficiency ε is defined as follows:

$$\varepsilon = \frac{S(K)}{K} , \tag{9}$$

where $S(K)$ stands for the speedup, and K is the number of processors. In this paper, we adopt the scaled speedup which is given in [14] to evaluate the parallel efficiency as follows:

$$S(K) = P_s + P_p K , \tag{10}$$

where P_s and P_p represent the fraction of the program which is performed in serial and parallel, respectively.

In order to investigate the parallel performance of ESGNNs, we apply to three classification problems (MONK's [7]) and the time series prediction (Mackey-Glass time series [8]). Next, we describe the brief explanation of these problems.

6.1 Classification Problems

MONK's problems [7] are widely used as the benchmark problems. The learning task of the MONK's problems is a binary classification task. Table 2 shows six discrete attributes of MONK's problems. Each of them is given by the following logical description of a class.

- Problem M_1: (head_shape = body_shape) or (jacket_color = red). From 432 possible examples, 124 were randomly selected for the training set. No noise was present.
- Problem M_2: Exactly two of the six attributes have their first value. From 432 examples, 169 were selected randomly. No noise was present.

Table 2. Six abilities of the MONK's problems

x_1: head_shape $\in$ round,square,octagon;
x_2: body_shape $\in$ round,square,octagon;
x_3: is_smiling $\in$ yes, no;
x_4: holding $\in$ sword,balloon,flag;
x_5: jacket_color $\in$ red,yellow,green,blue;
x_6: has_tie $\in$ yes,no;

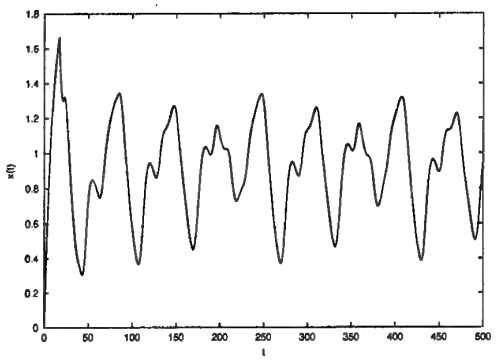


Fig. 5. Mackey-Glass time series from $x(0)$ to $x(500)$

- Problem M_3: (Jacket_color is green and holding a sword) or (jacket_color is not blue and body_shape is no octagon). From 432 examples, 122 were selected randomly. And among them there were 5% misclassification, i.e. noise in the training set.

6.2 Time Series Prediction

We generate chaotic time series from the differential equation of Mackey-Glass [8] which is used by many researchers as follows:

$$\frac{dx(t)}{dt} = -bx(t) + a\frac{x(t-\tau)}{1+x(t-\tau)^{10}}\ , \tag{11}$$

with $a = 0.2$, $b = 0.1$, $x(0) = 0.0$, and $\tau = 17$. The input time-lagged vector $\boldsymbol{x}_i$ is $(x_t, x_{t-m}, x_{t-2m}, x_{t-3m})$, and the desired output y_i is x_{t+L}. In this paper, m is 6 and L is 85. The training data set is used from t=0 to t=500 (Fig 5). Table 3 shows the relation between the input data $\boldsymbol{x}_i$ and the desired output y_i on training data set. The data from t=10000 to t=20000 are used for testing.

Table 3. Relation between the input data $\boldsymbol{x}_i$ and the output data y_i on training data set

i	$\boldsymbol{x}_i$	y_i
1	$(x_{18}, x_{12}, x_6, x_0)$	x_{103}
2	$(x_{19}, x_{13}, x_7, x_1)$	x_{104}
3	$(x_{20}, x_{14}, x_8, x_2)$	x_{105}
⋮	⋮	⋮
482	$(x_{499}, x_{493}, x_{487}, x_{481})$	x_{584}
483	$(x_{500}, x_{494}, x_{488}, x_{482}))$	$x_{585})$

7 Experimental Results

Fig. 6, Fig. 7, and Fig. 8 show the influence of the number of processors on misclassification rate (%) for MONK's problems M_1, M_2, and M_3 respectively. Misclassification rates are improved by computing the ensemble averaging of various SGNTs for all problems. On an average, these misclassification rates are improved respective 5.7%, 1.8%, and 4.1% in the case of the K is 10, and respective 6.7%, 2.1%, 4.4% in the case of the K is 30, for M_1, M_2 and M_3. The performance of the improving classification accuracy is saturated over 20 processors for all problems.

In the time series prediction, the influence of the number of processors on ARV for the Mackey-Glass time series is shown in Fig. 9. The result shows that the more the number of processors increases, the more ARV is improved. The improving efficiency is gradually saturated by increasing the number of processors same as classification problems.

The results of the bias/variance decomposition of MSE show some interesting associations between ARV and the number of processors. In order to illustrate that the tendency of the improvement of ARV, we compute the bias (B) and the variance (V) decomposition of MSE which are given by Eq.(6), Eq.(7) respectively for all cases (Fig. 10). We use the results of all 100 trials which are obtained from the same training data set D for evaluate B and V. Fig. 10 shows that though B is continue at the same level, V decreases. Hence, MSE and ARV decrease same tendency of V, and approximate B gradually in the case of the number of processors increases. Hence, the improvement of ARV is gradually saturated. It seems that the effect of the improvement of misclassification rate for classification problems is saturated for the same explanation as we have shown above.

Fig. 11 shows a part of prediction results between x_{15103} and x_{15203}[2] as an example of the relation between real time series and its predictions in the case of K is 1, 10, 30 respectively. Using the ensemble averaging of various SGNNs, the prediction output $f(\boldsymbol{x}_i)$ is approximated to the real Mackey-Glass time series y_i in almost cases.

Table 4 shows the computation time (CT, in sec.) and the speedup ($S(K)$) for classification problems (M_1, M_2, M_3), and time series prediction of the Mackey-Glass time series (MG). Here, the computation time is the total processing time of 100 trials. The results show that the parallel efficiency is approximately obtained the liner speedup for all problems, and the results of the time series prediction is a better speedup than three classification problems.

It is concluded that our method could improve the generalization capability by allocating each of SGNTs to each of processors, go on maintaining the high speed processing property of the single SGNN.

[2] They are taken in the time interval between $t = 15000$ and $t = 15118$ as input test data.

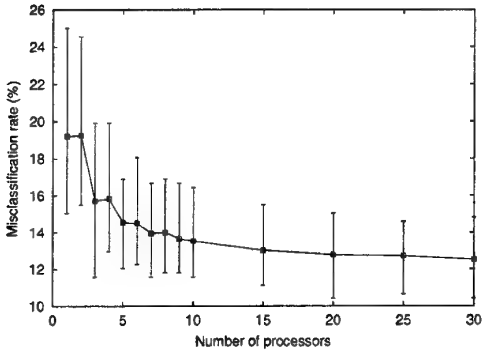


Fig. 6. Influence of the number of processors on misclassification rate (%) for M_1

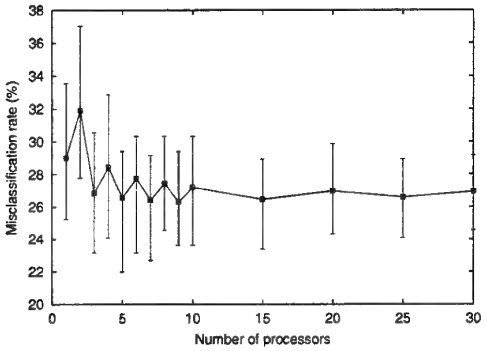


Fig. 7. Influence of the number of processors on misclassification rate (%) for M_2

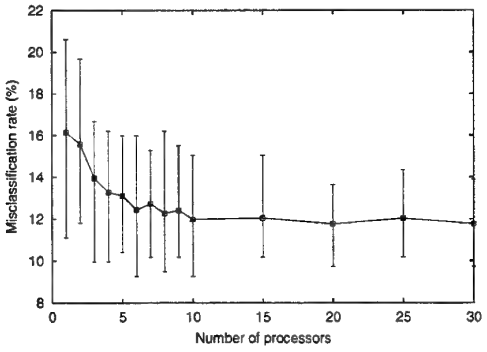


Fig. 8. Influence of the number of processors on misclassification rate (%) for M_3

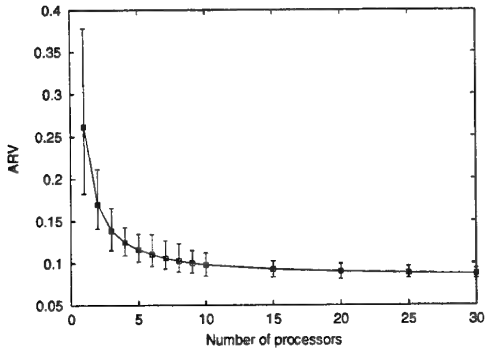


Fig. 9. Influence of the number of processors on ARV for the Mackey-Glass time series

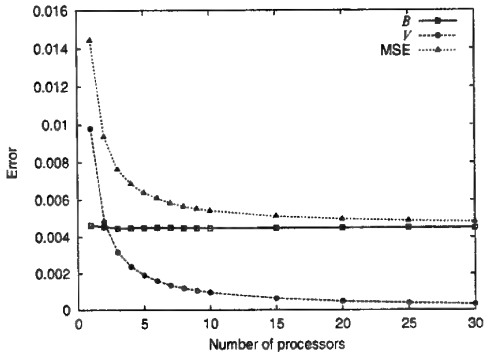


Fig. 10. Relation between error (B, V, and MSE) and the number of processors for the Mackey-Glass time series

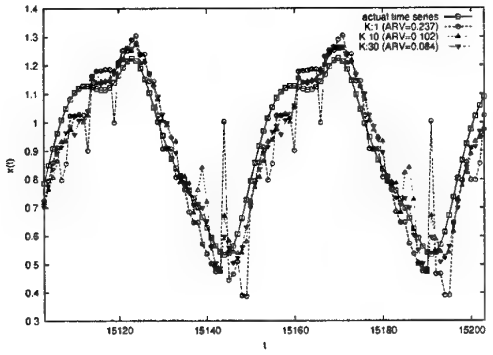


Fig. 11. A part of prediction results for the Mackey-Glass time series ($K = 1, 10, 30$)

Table 4. Computation time (CT), and speedup $S(K)$ as a function of the number of processors K for M_1, M_2, M_3 and the Mackey-Glass time series (MG)

	M_1		M_2		M_3		MG	
K	CT (sec.)	$S(K)$	CT (sec.)	$S(K)$	CT (sec.)	$S(K)$	CT (sec.)	$S(K)$
1	13.08	1.00	15.66	1.00	13.46	1.00	256.43	1.00
2	13.44	1.92	16.09	1.89	14.33	1.92	262.26	1.87
3	13.69	2.81	16.36	2.74	14.89	2.80	266.85	2.72
4	13.85	3.68	16.53	3.58	15.41	3.66	262.82	3.61
5	13.96	4.55	16.76	4.41	14.44	4.52	264.01	4.47
6	14.05	5.40	16.80	5.23	14.56	5.36	263.48	5.36
7	14.10	6.24	16.86	6.04	14.61	6.20	263.87	6.23
8	14.18	7.13	16.90	6.87	14.75	7.02	263.86	7.09
9	14.31	7.91	17.03	7.58	14.83	7.82	265.54	7.91
10	14.38	8.72	17.10	8.41	14.91	8.63	265.12	8.79
15	14.58	12.82	17.34	12.35	15.10	12.67	266.64	13.07
20	14.70	16.87	17.54	16.23	15.22	16.71	267.38	17.32
25	14.88	20.84	17.56	20.12	15.29	20.69	268.18	21.54
30	14.93	24.97	17.61	24.02	15.34	24.84	268.65	25.80

8 Conclusions

In this paper, we presented the parallel performance of ESGNNs for classification problems and time series prediction on the MIMD parallel computer. From the experimental results the following conclusions can be drawn:

- The improvement of the classification/prediction accuracy is expected by using various SGNTs which are allocated processors on the MIMD computer.
- The parallel efficiency is approximately obtained linear speedup for all problems.

Acknowledgements

The authors would like to thank the referees for their helpful comments and the Information Processing Center in Okayama University of Science for using the Paragon.

References

1. Wen, W. X., Pang, V. and Jennings, A.: Self-Generating vs. Self-Organizing, What's Different? In: Simpson, P. K. (eds.): Neural Networks Theory, Technology, and Applications. IEEE, New York (1996) 210–214
2. Kohonen, T.: Self-Organizing Maps. Springer-Verlag, Berlin (1995)
3. Wen, W. X., Jennings, A. and Liu, H.: Learning a Neural Tree. In: Int. Joint Conf. on Neural Networks. Beijing (1992) 751–756

4. Inoue, H. and Narihisa, H.: Performance of Self-Generating Neural Network Applied to Pattern Recognition. In: 5th Int. Conf. on Information Systems Analysis and Synthesis, Vol. 5. Orlando, FL (1999) 608–614
5. Rumelhart, D., Hinton, G. E. and Williams, R. J.: Learning Internal Representations by Error Propagation. In: Rumelhart, D., McClelland, J. and the PDP Reserch Group (eds.): Parallel Distributed Processing: Explorations in the Microstructure of Cognition. MIT Press, Cambridge, MA (1986) 318–362
6. Inoue, H. and Narihisa, H.: Improving Generalization Ability of Self-Generating Neural Networks through Ensemble Averaging. In: The Fourth Pacific-Asia Conf. on Knowledge Discovery and Data Mining. Lecture Notes in Computer Science, Vol. 1805, Springer, Berlin (2000) in press.
7. Thrun, S. B. et al.: The MONK's Problems — A Performance Comparison of Different Learning Algorithms. Technical report CMU-CS-91-197. Carnegie Mellon University (1991)
8. Mackey, M. C. and Glass, L.: Oscillation and Chaos in Physiologist Control Systems. Science **197** (1977) 287–289
9. Weigend, A. S., Huberman, B. A. and Rumelhart, D. E.: Predicting the future: A connectionist approach. Int. J. of Neural Systems **1** (1990) 193–209
10. Fisher, D. H.: Knowledge Acquisition via Incremental Conceptual Clustering. Machine Learning **2** (1987) 139–172
11. Haykin, S.: Neural Networks: A comprehensive foundation. Prentice-Hall, Upper Saddle River, NJ, second edition (1999) 353–355
12. Naftaly, U., Intrator, N. and Horn, D.: Optimal Ensemble Averaging of Neural Networks. Network **8** (1997) 283–296
13. Geman, S., Bienenstock, E. and Doursat, R.: Neural Networks and the Bias/Variance Dilemma. Neural Computation **4** (1992) 1–58
14. Pardalos, P. M., Phillips, A. T., Rosen, J. B.:Topics in parallel computing in mathematical programming. Science Press, New York (1992) 5–6

An environment to learn concurrency

Giuseppina Capretti, Maria Rita Laganà and Laura Ricci

Università degli Studi di Pisa, Dipartimento di Informatica, Corso Italia 40,
56125 Pisa ITALIA
{*capretti, lagana, ricci*}@di.unipi.it

Abstract: The paper describes the Orespics system, a tool defined to learn the basic concepts of concurrency. Orespics defines an imperative language by putting together the primitives of the Logo language turtles and a set of concurrent constructs. We show that the system may be employed to plan didactic trainings in order to teach the basic concepts of concurrent programming: in particular we show a course to learn the semantics of different types of communication primitives.

1 Introduction

Recently, the educational challenge of teaching parallel programming has acquired great importance: our teaching experience shows that students find several difficulties in understanding and learning the concurrent paradigm even if the world where we live and work is naturally concurrent.

We think that this situation is due to the way in which the concurrent paradigm is usually presented in our high education courses, where students have their first experience in concurrency through the programming of operating systems or of complex scientific applications.

We believe that this experience should be acquired at an earlier age, through simpler and friendlier programming environments. Our idea is to define an environment where the students may create *micro-worlds*, i.e. virtual worlds populated by creatures interacting through the exchange of messages.

We have chosen the paradigm of message exchange because we believe that it simulates naturally the anthropoids communication. Futhermore single system including both the message passing paradigm and the shared memory one is feasible but not suitable for didactic purposes. Some didactic principles establish that the learning is more valid if it follows a sequence of consolidated steps, where each step corresponds to the acquirement of a well defined concept. According to them, we avoid to mix the message passing paradigm with the shared one in a single didactic environment because this could introduce confusion in the learning process. Furthermore no standard definition of the shared memory paradigm has been given till now: the proposed models differ one another for the adopted consistency model [9].

The system we propose may be considered an agent one [16], where each agent is a process and is programmed through a language integrating the Logo [6] turtles movement primitives in an imperative concurrent language: the students may analyse the interactions of processes through an inspection of the evolution of the virtual creatures on the screen.

As our fundamental choice, the language we have defined is very close to Pascal and the communication primitives are stripped out the versions of the MPI library [11] so the knowledge acquired through our system may be useful in the students' future life. On the other hand, as stated above, it is difficult to present, as a first approach, a professional language (C language) extended with MPI primitives because the students get bored because of the technical details without significant conceptual added value.

The most important didactic characteristic of our system is the identification between the agent and the character of the virtual world: this introduces a natural transformation from an abstract concurrent problem (e.g. multiple readers – single writer problem) into a "concrete" situation (the example of the ants and the corn in the section 5 Experiment). The system visualises the evolution of the concurrent processes through the evolution of the world: this kind of visualisation is deeply different from that of the classical debuggers for parallel programs [7, 10].

The theory underlying our proposals is the constructionism of Papert [12]. According to this paradigm, learning is an active process: the students build their mental infrastructures through a free exploration of the world.

The first embryonic version of our system, presented in [5], is evolved in [2, 3, 4]; the last version of the system is presented in this paper. The primitives introduced in the different versions of the system, has enabled the definition of different didactic trainings characterised by increasing levels of difficulties whose aim is to propose the resolution of more and more sophisticated concurrency problems.

In particular, the didactic training we describe in this paper shows how the same problem may be programmed through a sequence of solutions characterised by an increasing degree of agents' autonomy: the increase of autonomy corresponds to the use of communication primitives with higher degrees of non determinism; furthermore, the students learn a decentralised way of thinking by exploiting the functionality of the system.

The paper is organised as follows: firstly we present an analysis of some of the existing animate systems based on concurrent paradigms, then we introduce our new system, Advanced Orespics, and its Orespics-PL language; finally we propose an example of didactic training to teach how non-determinism increases autonomy.

2 Related work

Any programming environment designed to support the building of an animate system should offer some basic world-modelling capabilities and present them in an easy and accessible form. It must support the simultaneous animation of multiple objects. It must support object autonomy: i.e. objects should operate under their own control and must be able to sense the surrounding environment as well as interact with other animate objects.

Several interesting proposals of animate system are known today.

StageCast (en evolution of the KidSim system [15]) is a simulation system of micro-worlds developed by Apple's Advanced Technology Group. Students who use this instrument may create the characters of a micro-world and program their behaviour through a set of rules. Each object of the micro-world is identified by three properties:

- the appearance, that describes how the objects appear on video in a particular context,
- the rules, which define its behaviour in a particular context,
- the property, that allows to store a particular event.

A student that uses StageCast creates the prototypes of the characters, inserts several copies of each of them in the micro-world, and activates the simulation of the world. At every tick (that represents the tick of the StageCast clock) each object of the world examines its context and decides the new state by looking for its set of rules. The creation of rules is made *by demonstration*, defining the relationships existing between two graphic contexts: by using the mouse, a student arranges the objects of the world defining the initial context of application of the rules (*before* context) and then he/she modifies their disposition in the *after* context. The execution of the world is a change in the disposition of objects.

When the simulation is active, all the objects on the screen move in parallel. This is a simulated concurrency. The system applies the rules of transition following the order of insertion of the characters in the world. It applies the rule of transition to the first object of the world and evaluates the second one in the context eventually modified by the application of the rules in sequence, starting from the first one.

StarLogo [14] is a programming environment to explore the behaviour of decentralised systems developed at M.I.T.. Through this system, a student may program and control, in parallel, hundreds of turtles through a Logo-like programming language. The world of the turtles is alive: it is composed by hundreds of patches that may be considered like turtles that are programmable but without movement. Turtles move in parallel and use the patch to exchange messages. No mutual exclusion is guaranteed on the patch and no explicit use of concurrent constructs is needed to exchange messages.

ToonTalk [17] is a concurrent object oriented programming language based upon the concurrent constraint programming paradigm and is based on an animated syntax and programming environment. The concurrency and the message exchange paradigm are limited and the target of the system is limited to children.

All these systems do not allow a clear and/or complete definition of communication among concurrent entities.

The **Orespics** system, developed at the Computer Science Department of the University of Pisa [1, 5] is programmable with Logo-PL language, a local environment language in which students guide the action of at most 8 virtual agents.

Logo-PL has control flow, movement and communication commands and expressions. The Logo-PL language defines a set of communication primitives. In particular, the prototype version implements basic primitives to send and receive messages. The receive primitive is synchronous and asymmetric while the send primitive is synchronous and symmetric [8]. The introduction of these primitives allows students to subordinate the actions of certain creatures to the action of others. As clearly shown, in this prototype, the set of the communication primitives is extremely poor.

3 The Advanced Orespics system

Advanced Orespics is a project we are developing as further improvement of the Orespics system described above. Its programming language is called Orespics-PL and is based on the local environment model and on the explicit use of the communication primitives. The Advanced Orespics system has substituted the previous one because a richer set of communication primitives is defined, activation and termination constructs are introduced and no limit to the number of interacting actors in the world is imposed. Each actor is an agent of an animate system and has the attributes of autonomy, purposefulness and the ability to react to the surrounding environment by the exchange messages paradigm. An agent is characterised by a set of properties: the initial position on the screen, its appearance and the code of its program.

The system gives the users an interface to define all these properties. The system has a set of pre-defined fantastic and real characters like aliens and animals. The students may choose the most suitable character according to the situation to solve.

The sequential part of Orespics-PL includes traditional imperative sequential constructs (repeat, while, if ...) and all turtle primitives of the Logo language [6]. Orespics-PL language offers all the elementary data types (integer, boolean..): the only data structure is the list. Some of the operations defined on list type are *getFirst(list), first(list)* and *second(list)*: *getFirst* returns the first item of *list* and pops it up; *first* and *second* return respectively the first and the second ones and do not pop them up.

The set of primitives, functions and procedures used in the following examples are:

- *versus(x, y)*, which returns the direction to assume to reach the point of co-ordinates *x* and *y*,
- *distance(x, y)*, which returns the distance between the position of agent and the point *(x, y)*,
- *set_heading(angle)*, which turns the agent in the direction given by the *angle*,
- *jump(x, y)*, the agent jumps to the point of co-ordinates *(x, y)*,
- *show(s)*, the agent shows on the screen the string *s*,
- *random(val)*, which returns a random value included in +/- *val*. If parameter is zero, it returns a random value according to the common definition.

As regards the concurrent part, the new language defines the following types of primitives:

- termination and activation,
- synchronous and asynchronous send and receive,
- broadcast/multicast send,
- asymmetric receive,
- a command to activate a population of agents.

We present, in the following, the syntax and semantics of these constructs.

The behaviour of an agent is specified between the keyword *Agent* and *end* where the word that follows *Agent* is its name.

We now describe all the communication primitives of the language. As far as semantics is concerned, that the communication partner is active or it has finished the program execution or it is not ready to accept/receive the message is irrelevant.

The syntax of the synchronous symmetric primitives is:

send&wait *msg* **to** *agent*
wait&receive *var* **from** *agent*

The semantics of synchronous and asynchronous primitives is well known in literature [8]. With regards to the synchronous primitives, when an agent sends a message to another one and its partner is not ready for communication or is not active, it waits until the message has been received. The semantics of synchronous receive primitive is analogous. The use of these primitives is shown in the following example.

Example 1

Consider the following problem:

"An agent wants to sleep for a time slice, after which it want to be woken up by an alarm clock"

This micro-world may be programmed defining two agents. The first one sends the other one the amount of time it wants to spend sleeping and waits till the other one wakes it up. The second agent receives the message and simulates an alarm clock.
It is worth noticing that the behaviour of the sleepy agent may be easily simulated exploiting a synchronous receive primitive to make it wait for the alarm.

```
Agent Clock
    receive&wait timeSlice from Sleepy;
    tick←0;
    repeat  tick ← tick + 1;
    until (tick = timeSlice)
    send&wait timeSlice to Sleepy
end
```

```
Agent Sleepy
    send&wait timeSlice to Clock;
    receive&wait timeSlice from Clock;
    show "Get up!";
end
```

The syntax of the asynchronous primitives is:

receive&no_wait *var* **from** *agent*
send&no_wait *msg* **to** *agent*

As for the asynchronous primitives, an agent sends/receives a message to/from another one, but it does not wait for the successful issue of the communication. When an agent executes a *send&no_wait*, it does not wait for the receiver to get the message and it goes on with its execution. If the receiver is not ready to accept it or it is not active, the message is inserted in a queue where messages are inserted and taken according to the order of arrival. We suppose that messages sent by one agent to another one are received in the same order as they are sent. When an agent executes a *receive&no_wait*, it checks the existence of some incoming messages and goes on. If the queue is empty, the message has no meaning, and no value is assigned to the *var*. The meaning of *var* may be checked through the function *in_message()* which returns a **true** value if the last executed *receive&no_wait* has picked up a valid message, and a **false** value otherwise.

A process executing the *receive&no_wait* performs a non-deterministic choice: we suggest that a suitable introduction of non-determinism in concurrent programs should be given when this primitive is introduced to the students.

We have defined synchronous and asynchronous broadcast/multicast send. The syntax of the synchronous broadcast send is:

sendAll&wait*msg*

The syntax of the synchronous multicast send is:

sendAll&wait*msg***to** *list_agents*

In the first case an agent sends a message to all agents while in the second case it sends a message only to the subset of agents defined in *list_agents*in both cases it waits for the successful issue of all the communications. Its execution is suspended until all the receivers get the message.

We also define asynchronous broadcast and multicast primitives: as regards the syntax, it is sufficient to substitute the word **wait** with **no_wait** and the semantics is analogous to the symmetric case.

The language includes synchronous and asynchronous asymmetric receive. The syntax of the synchronous asymmetric receive is:

receiveAny& wait*msg***from***list_agents*
receiveAny& wait*msg*

In the first case an agent receives a message from any of the active agents in *list_agents*, while, in the second case, it receives a message from any active ones, and in both cases, it waits until one message has been received. If more than one message arrives, its selection is non-deterministic.

In a system allowing the simulation of micro-worlds, a command to specify several agents with the same behaviour is useful. To create a population of agents with the same behaviour each agent's code has to be included between the keywords *GenericAgent* and *end*.

The syntax of the construct is

newAgent(*1..N*) agentName

The system generates the codes of N agents, which are called agentName1, agentName2, ..., agentNameN.

The usage of broadcast send, asymmetric receive and newAgent command are shown in the following example.

Example 2

"The coach of the Chicago Bulls has to select the pivot man from a set of basket ball players."

```
Agent Coach
    receiveAny&wait max;
    for (i←1 to 4)
        receiveAny&wait height;
        if (max < height)
            max←height;
        endif
    endfor
    sendAll&no_wait max;
end
```

```
GenericAgent BasketBall_Player
    send&no_wait my_height to Coach;
    receive&wait max from Coach;
    if (max = my_height)
        forward 10;
    endif
end
```

The *Coach* agent receives the height of each player, computes the maximum one and tells it to all players. The one having that height moves forward.

In this example we use newAgentconstruct to create the players of the team as follows:

newAgent(1..5) BasketBall_Player

This construct creates 5 players: *BasketBall_Player1,...., BasketBall_Player5.* ◇

4 A didactic training to learn the communication semantics

This section shows how a didactic training to teach communication semantics may be planned. We introduce an example of didactic training, whose goal is to allow the students to learn the semantics of different communication primitives and to let them understand how the use of non-deterministic primitives increases the autonomy of the agents.

According to the constructionist approach, this is obtained by proposing a set of proper problems to the students and letting them solve these problems in the way they prefer. The whole process is obviously guided by the teacher; nevertheless the student may try different solutions and verify the effects of his/her program directly on the screen in case s/he changes the program.

We shall show only the code of the agents used to solve the proposed problems: we suppose they have just been created and that each of them has been properly defined.

The didactic training proposes a sequence of examples where the agents are characterised by an increasing degree of autonomy: this implies the use of the non-deterministic primitives.

We propose the following situation:

> *"In a field there are two ants called Z ant and T ant. They are searching for food and make an agreement: the first who finds it notifies the other one with the position of the food. "Good luck T". "Good luck Z. Let us begin"*

First, we propose the students to solve the problem by using only deterministic send and receive commands. In this case, the solution will be characterised by a strict synchronisation between the agents implementing the two ants: for instance, Z and Q ants may exchange, at each step, a message notifying other if it has found the food or not. If the agent has found it, it sends its co-ordinate to the other and stops moving. All

send primitives we use in this version are asynchronous to avoid the deadlock, and the *receive* ones are synchronous.

```
Agent Z_ant
    x ← random();
    y ← random();
    jump(x, y);
    l_found← false;
    You_found←false;
    repeat
        x ← random(25);
        y ← random(15);
        right x;
        forward y;
        if here_food(myX, myY)
        then
            send&no_wait "Found" to T_ant;
            send&no_wait [myX, myY] to
            T_ant;
            l_found← true;
        else
            send&no_wait "Not Found" from
            T_ant;
        endif
        receive&wait [x] from T_ant;
        if (x = "Found")
            receive&wait [x, y] from T_ant;
            You_found←true
        endif
    until (l_found OR You_found);
    if You_found
    then
        set_heading (versus(x, y));
        forward distance(x, y);
    endif
end
```

```
Agent T_ant
    x ← random();
    y ← random();
    jump(x, y);
    l_found← false;
    You_found←false;
    repeat
        x ← random(25);
        y ← random(15);
        right x;
        forward y;
        if here_food(myX, myY)
        then
            send&no_wait "Found" to Z_ant;
            send&no_wait [myX, myY] to
            Z_ant;
            l_found← true;
        else
            send&no_wait "Not Found" from
            Z_ant;
        endif
        receive&wait [x] from Z_ant;
        if (x = "Found")
            receive&wait [x, y] from Z_ant;
            You_found←true;
        endif
    until (l_found OR You_found);
    if You_found
    then
        set_heading (versus(x, y));
        forward distance((x, y));
    endif
end
```

In the second version, described here below, we propose students to employ non-deterministic primitives to increase the autonomy of each agent: each agent does not know the behaviour of the other one, in particular it does not know if and when the other one is ready to communicate.

In this case, the agent has to use a communication primitive which allows to check if a message is incoming from the environment without stopping its execution: this is obtained through *receive&no_wait*.

```
Agent Z_ant
    I_found← false;
    You_found← false;
    x← random();
    y← random();
    jump(x, y);
    repeat
        x← random(25);
        y← random(15);
        right x;
        forward y;
        if here_food(myX, myY)
        then
            send&no_wait [myX, myY]
            to T_ant;
            I_found←true;
        else
            receive&no_wait [x, y] from
            T_ant;
            if in_message()
            then
                You_found←true;
            endif
        endif
    until (I_found OR You_found);
    if You_found
    then
        set_heading (versus(x, y));
        forward distance((x, y));
    endif
end
```

```
Agent T_ant
    I_found← false;
    You_found← false;
    x← random();
    y← random();
    jump(x, y);
    repeat
        x← random(25);
        y← random(15);
        right x;
        forward y;
        if here_food(myX, myY)
        then
            send&no_wait [myX, myY]
            to Z_ant;
            I_found←true;
        else
            receive&no_wait [x, y] from
            Z_ant;
            if in_message()
            then
                You_found←true;
            endif
        endif
    until (I_found OR You_found);
    if You_found
    then
        set_heading (versus(x, y));
        forward (distance(x, y));
    endif;
end
```

Each ant moves randomly, checking for the presence of the food. If an ant finds it, it sends the other one the food co-ordinates and stops moving. If it does not find it, it checks the presence of an incoming message from the other ant. If no message is present it goes on moving, otherwise it receives the message and reaches the food. The evolution of the program is shown in fig. 1.

It is important to stress that in this case the agents are completely autonomous. Each one may be programmed without knowing the behaviour of the other agent: this is the main difference between this example and the previous ones. The increase of autonomy is obtained through the use of the *receive&no_wait* primitive that allows each agent to perform a non-deterministic choice whose result depends on its interaction with the surrounding world.

Finally, we propose a generalisation of the previous problems. A population of ants is involved and more food sources are present.

Each ant moves randomly on the screen, and if it finds the food, it lets all the other ones know. Otherwise, it checks if any ant has found the food: since more than one source of food is present, each ant may receive more than one notification from the other ones. In this case it reaches the nearest source of food.

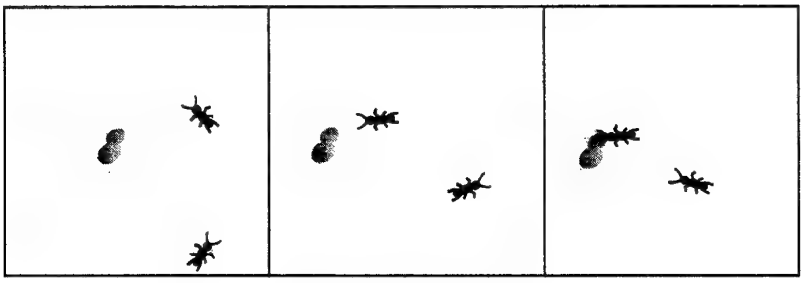


Fig. 1 Evolution of the movement

```
GenericAgent Ant
    best_distance◄- MaxDistance;
    best_x◄-MaxX;
    best_y◄-MaxY
    I_found◄-false;
    You_found◄-false;
    x ◄- random();
    y ◄- random();
    jump(x, y);
    repeat
        angle◄- random(25);
        far◄- random(15);
        right angle;
        forward far;
        if here_food(myX, myY)
        then
            sendAll&no_wait [myX, myY];
            I_found◄-true;
        else
            repeat
                receiveAny&no_wait msg;
                if in_message ()
                then
                    You_found◄-true;
                    if (distance(x, y) <
                    distance(best_x, best_y))
                        best_x ◄- x;
                        best_y ◄- y;
                    endif
                endif
            until(in_message())
        endif
    until (I_found OR You_found);
    if You_found
    then
        set_heading (versus(best_x, best_y));
        forward distance(best_x, best_y);
    endif
end
```

The behaviour of each ant is similar to that of the previous examples: if an ant finds the food it exploits a broadcast send (*sendAll&no_wait*) to inform all others. The send is asynchronous because no strict synchronisation is required. If an ant does not find the food, it checks, through the function *in_message()*, for the presence of any incoming messages: it is worth noticing that the agent exploits the *receiveAny&no_wait* because it does not know in advance which ant has found the food. A further level of non-determinism is present since the ant does not know who has sent the message. All incoming messages are picked up and the message containing the nearest co-ordinates is selected while others are discarded: the ant then reaches the point corresponding to the selected message.

With the command

```
newAgent (1..5) Ant
```

we create several ants with the same behaviour which move randomly on the screen and search for the food. The newAgentcommand, according to its semantics, generates the code of five ants, Ant1,..., Ant5. The use of broadcast send and asymmetric receive makes each ant independent from the other: its behaviour is not bound to a particular ant.

The following figure shows what happens in the case that two ants find the food at the same time.

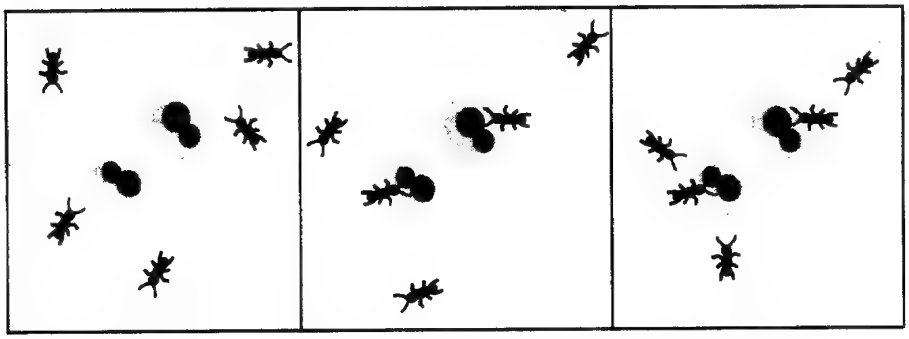


Fig. 2 Population of ants: evolution of the movement

5 Experiment

The last version of the Orespics system [13], has been employed to carry out a preliminary experiment. The target of it has been a fifteen years old student, called Massimiliano, who is currently studying Pascal and with some experience with Logo. We are happy to show the positive result of this experiment.

Massimiliano has rapidly learnt the basic features of the system and, when required to define a micro-world, he has proposed a famous game in Italy "Worms II". Since this problem offers a low degree of parallelism, we have proposed him an alternative one, a deterministic version of the classical "multiple readers/single writer" problem, which we have proposed in the following way:

"A tractor brings some corn into a basket where some ants are eating. To avoid ants' death, the tractor can not unload the corn when the ants are eating. The ants eat all the corn in the basket before to leave it and the tractor fills it completely with the corn".

Massimiliano has immediately defined the agents of this micro-world: the most surprising issue was the introduction of an animated basket. In the concurrency framework, this corresponds to a manager for the shared resource, i.e. the corn. The animated basket co-ordinates the activities of the ants and of the tractor by alternating their accesses to the basket.

The kind of the messages employed show the age of the boy: for instance, the tractor sends to the basket the message *"levatemele di torno"*(*"get rid of the ants"*) to drive out the ants, each ant sends the message "*che paura!*"(*"I am afraid!"*) as soon as it has gone out of the basket.

We have noticed that the boy has always preferred the synchronous communication primitives to the asynchronous ones because "*In this case it's the same*".

Finally, the only relevant error of the program is a typical "time dependent" bug: the student has exploited a "wait 15" instruction in the agent "basket" to define the time the tractor needs to unload the corn. This does not guarantee the mutual exclusion of the accesses to the basket. We have shown him that it is guaranteed only if the end of

the unload operation is notified from the tractor agent to the basket one through an explicit message.

Fig.3 shows some snapshots of the micro-world.

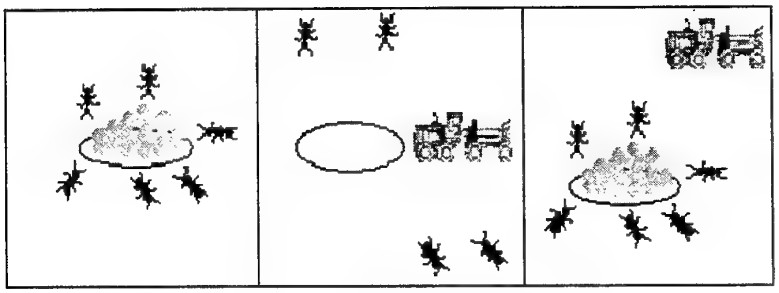


Fig. 3 Snapshots of the microworld

6 Conclusions

We may create lots of new examples in which the agents co-ordinate and synchronise themselves. Classic problems like the game of life, the simulation of a biological system, and so on may be naturally realised in our system.

We have studied the possibility of creating several typologies of agents characterised by a richer set of personal properties, for example, in the case of the ants we give them the ability to move the antennae or in the case of the dog the ability to bark. We are implementing a version in which the character may be created or imported by the student.

7 Reference

[1] G. Capretti, Strumenti per l'apprendimento della concorrenza nella didattica dell'informatica, Master of Computer Science Thesis, Computer Science Department, University of Pisa, December 1997

[2] G. Capretti, A. Cisternino, M. R. Laganà and L. Ricci, A concurrent microworld, ED-MEDIA 99 World Conference on Educational Multimedia, Hypermedia & Telecommunications, Seattle, Washington, June 19-24th

[3] Giuseppina Capretti, Maria Rita Laganà, Laura Ricci – Decentralised programming of communicating *turtles* – EUROLOGO 99 - Sofia, Bulgaria

[4] Giuseppina Capretti, Maria Rita Laganà, Laura Ricci, Learning concurrent programming: a constructionist approach, PaCT 99 Parallel Computing Tecnologies - San Pietroburgo, Russia,

[5] G. Capretti, M. R. Laganà and L. Ricci, Micro-world to learn concurrency, SSCC'98, 22-24 September 1998, Durban, South Africa, 255-259

[6] B. Harvey, Computer Science Logo Style, The Mit Press, Cambridge, 1997

[7] P. Kacsuk, J.C. Cunha, G. Dozsa, J. Lourenco, T. Fadgyas, T.Antao, A Graphical Development and Debugging Environment for Parallel Programs,. Parallel Computing, 22, 1997, pp. 1747-1770,

[8] C. A. R. Hoare, Communicating Sequential Process, Comm. of the ACM, Vol. 21, No. 8, Aug. 1978, pp. 666-677

[9] V.Milutinovic e P,Stenstrom, Proceedings of the IEEE Special issue on Distributed Shared Memory systems, vol. 87, n. 3, March 1999, pp. 397-532.
[10] C.E.Mcdowell, D.P.Helmbold, Debugging Concurrent Programs ACM Computing Surveys, Vol.21, No. 4, December 1989, pp. 593-622
[11] P.S. Pacheco Parallel programming with MPI, Morgan Kaufmann, 1977
[12] S. Papert, Mindstorm: children, computer and powerful ideas, Basic Books, New York, 1980.
[13] S. Puri, Orespics: un ambiente per l'apprendimento della concorrenza, Master of Computer Science Thesis, Computer Science Department, University of Pisa, May 2000
[14] M. Resnick: Turtles, termites and traffic jam: exploration in massively parallel micro-world, The MIT Press, Cambridge, 1990.
[15] D. C. Smith and A. Cypher, KidSim: end users programming of simulation, Apple Computer Inc., 1997. http://www.acm.org/sigchi/chi95/Electronic/documnts/papers/ac1bdy.htm
[16] M. D. Travers, Programming with agents: new metaphors for thinking about computation, Bachelor of Science Thesis at Massachusetts Institute of Technology, 1996
[17] http://www.toontalk.com

Dynamic Load Balancing Model: Preliminary Results for Parallel Pseudo-Search Engine Indexers/Crawler Mechanisms using MPI and Genetic Programming

Reginald L. Walker

Computer Science Department
University of California at Los Angeles
Los Angeles, California 90095-1596
rwalker@cs.ucla.edu

Abstract. Methodologies derived from Genetic Programming (GP) and Knowledge Discovery in Databases (KDD) were used in the parallel implementation of the indexer simulator to emulate the current World Wide Web (WWW) search engine indexers. This indexer followed the indexing strategies that were employed by AltaVista and Inktomi that index each word in each Web document. The insights gained from the initial implementation of this simulator have resulted in the initial phase of the adaption of a biological model. The biological model will offer a basis for future developments associated with an integrated Pseudo-Search Engine. The basic characteristics exhibited by the model will be translated so as to develop a model of an integrated search engine using GP. The evolutionary processes exhibited by this biological model will not only provide mechanisms for the storage, processing, and retrieval of valuable information but also for Web crawlers, as well as for an advanced communication system. The current Pseudo-Search Engine Indexer, capable of organizing limited subsets of Web documents, provides a foundation for the first simulator of this model. Adaptation of the model for the refinement of the Pseudo-Search Engine establishes order in the inherent interactions between the indexer, crawler and browser mechanisms by including the social (hierarchical) structure and simulated behavior of this complex system. The simulation of behavior will engender mechanisms that are controlled and coordinated in their various levels of complexity. This unique model will also provide a foundation for an evolutionary expansion of the search engine as WWW documents continue to grow. The simulator results were generated using Message Passing Interface (MPI) on a network of SUN workstations and an IBM SP2 computer system.

1 Introduction

The addition of new and improved genetic programming methodologies [14],[36] will enable the preliminary Pseudo-Indexer model [33] to generate a population of solutions [6],[22] that provide some order to the diverse set of Web pages

comprising the current and future training sets. The applicability of genetic programming to this task results from the existence of an adequate population size in relation to the difficulty in organizing the diverse set of Web pages [30],[34].

Studies of parallel implementations of the genetic programming methodology [6],[15],[18],[25] indicated that population evaluation is the most time-consuming associated process. Population evaluations for the Pseudo-Search Engine's Indexer will result from calculating the fitness measures associated with each Web page after one of the following: 1) parsing the training set, 2) additions to the training set, or 3) the execution of one or more of the GP operators. The cost associated with the fitness computations [18] offsets the cost associated with the load balancing and communication overheads. The previous GP studies have resulted in dynamic load-balancing schemes which can be used to monitor the irregularity in processor work loads, a result of parsing variable size Web pages. A major shortcoming of GP applications [6] is the amount of execution time required to achieve a suitable solution.

2 Chromosome Modeling using Genetic Methodologies

2.1 Genetic Methodologies

Genetic programming is an evolutionary methodology [36] that extends the techniques associated with Genetic Algorithms (GAs) [4]. The evolutionary force of these methodologies reflects the fitness of the population. The basis of GAs results from designing an artificial chromosome of a fixed size that maps the points in the problem search space to instances of the artificial chromosome. The artificial chromosome is derived by assigning variables of the problem to specific locations (genes). The memes [1] denote the value of a particular gene variable. Genetic algorithms provide an efficient mechanism for multidimensional search spaces that may be highly complex and nonlinear. The components of a GP are:

1. Terminal set. The terminal set consists of input variables or constants.
2. Function set. The functional set varies, based on the GP application, by providing domain-specific functions that construct the potential solutions.
3. Fitness measure(s). The fitness measure(s) provide numeric values for the individual components associated with the members of a population.
4. Algorithm control parameters. The algorithm control parameters are dependent on population size and reproduction rates (crossover rate and mutation rate).
5. Terminal criterion. The terminal criterion uses the fitness measures to determine the appropriateness of each solution based on an error tolerance or a limit on the number of allowable generations.

The search space is defined by the terminal set, function set, and fitness measures. The quality and speed of a GP application is controlled by the algorithm control parameters and terminal criterion.

Genetic programming is composed of fitness evaluations [15],[18] that result from the application of genetic operators to individual members of a chosen

population (Web pages). The operators incorporated in this methodology are individual (or subgroup) migration, reproduction, crossover, and mutation. The use of a linear string of information can result from the direct modeling of DNA. The outcome from applying the genetic operations to this string of information corresponds to obtaining a globally optimum (or near-optimum) point in the original search space of the problem.

The migration operator consists of a process to select individual(s) to delete from the population. The reproduction operator consists of a process to copy an individual into a new population. The crossover operator generates new offspring by swapping subtrees of the two parents. The mutation operator randomly selects a subtree of a chosen individual. This process then randomly selects a new subtree to replace the selected subtree. The application of the mutation operator reduces the possibility of achieving a solution that represents a local optima. The selection of individuals from the distinct subpopulations may follow several formats. A new methodology will be developed when applying these operators to subpopulations of Web pages.

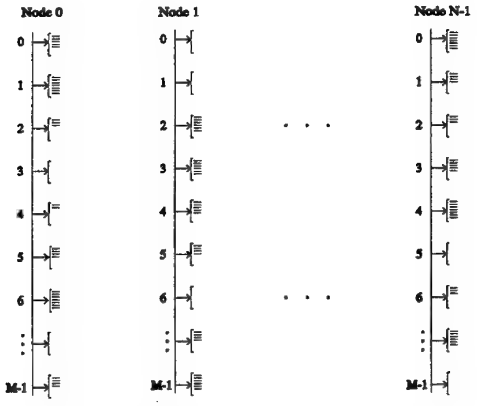


Fig. 1. Distribution of Web pages.

2.2 Modeling Chromosomes

The Pseudo-Search Engine Indexers' hybrid chromosome structure in Figure 1 follows the methodologies of GP and GAs. These structures represent subsets of Web pages (subpopulations) that reside at each node (Web site) in a distributed computer system. Each strand of genes that reside on each $Node_i$ (Web site) is viewed as a set of the genetic components of an individual member of a simulated species. Each horizontal strand in the chromosome structure represents a Web

page that would translate into a meme. The bracket to the left of the Web pages implies that the pages have similar characteristics to those that comprise a gene (allele) and its memes.

The components of the genes are referred to as the memes and the number of memes vary within each allele. The memes are the actual Web pages corresponding to primitive features that are contained at each Web site. New allele are formed by the addition of new Web pages at a given Web site. When new memes are added to enhance an alleles' current set of memes, each allele can grow in size, but its chromosome length remains fixed. The application of the GP crossover operator results in two new chromosomes, formulating from the transmission of components of the genetic makeup of the parents. The bracket mechanism provides a numerical order to the Web pages in this structure. This approach provides a mechanism to facilitate the evolution of diverse nodes. The use of a single population leads to panmictic selection [15] in which the individuals selected to participate in a genetic operation can be from anywhere in the population. The method used to avoid local optima [22] involves subpopulations [24]. This model will be expanded by simulating double-stranded RNA genomes [10] as the population of indexed Web pages grows.

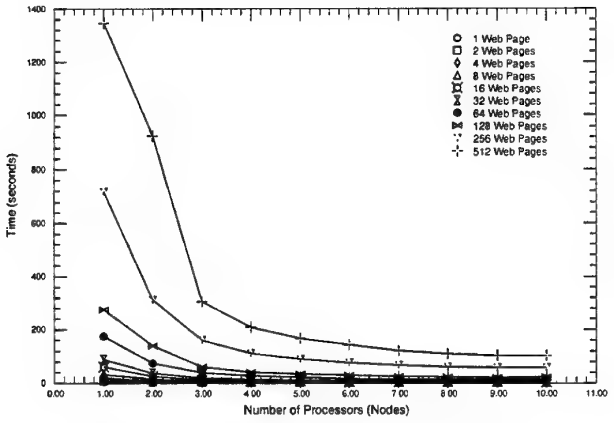


Fig. 2. Execution times for the workstations.

3 The Biological Model

The biological model for the Pseudo-Search Engine [32] is based on the social structure of honeybees [9],[29]. A mathematical model of the social structure of honeybees will be used to enhance the incorporation of GP methodologies [14]

since the bee colony represents a highly evolved biological system [17] which forms a basis to model the continuous expansion of Web pages.

The genetic programming approach incorporates the following genetic operators: migration, reproduction, cross-over, and mutations. A similar group of evolutionary operators for honeybee colonies are: migration, swarming, and supersedure (replacement of an existing, older queen by a younger queen following a fight). The evolutionary operators associated with the queen, drones, and worker bees are similar to the genetic programming operators. The cross-over operator is similar to the mating process for the queen bee, but differs since the parent chromosomes cease to exists in GP but only the queen persist in the evolutionary sense. The children chromosomes replace their corresponding parent chromosomes in standard GP. The migration operator in GP purges an existing subpopulation of the least desirable members (traits) and in some cases the best member (trait) [25]. This process was implemented in GP as an attempt to avoid local optimals.

In a true evolutionary model individuals migrate from one subpopulation to another [7] for many diverse reasons such as crowding, changes in the environmental conditions, limitations on colony activities, or members becoming disoriented. These external evolutionary factors benefit the gene pool by ensuring diversity. The mating ritual [9] for queen bees in colonies provides a built-in mechanism for incorporating a host of diverse genetic profiles into existing and/or new colonies. The drone bee mates once and dies – a process similar to a worker bee using its stringer and dying.

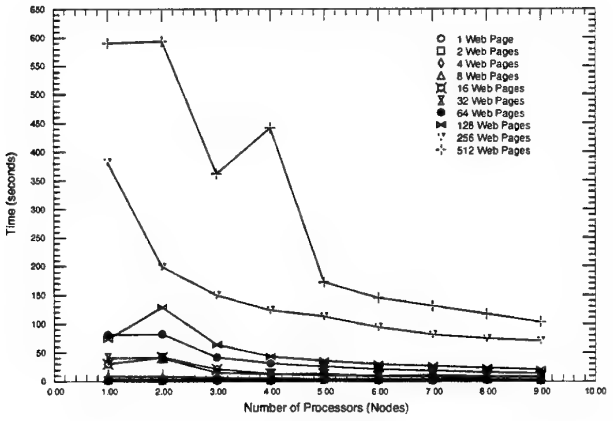


Fig. 3. Execution times for the IBM SP2.

4 Limitations of the Parallel Implementations

4.1 Timing Results

Message-passing studies [8],[26] have been conducted to determine the efficiency of parallel programs, implemented on shared as well as distributed memory computer systems. The implemented message-passing paradigm depended upon a client-server model [19] with $n - 1$ clients for a sub-cluster of n nodes as its basis. The message size used in all data transmissions was consistent and the data type contained an array of 101 characters. The sending and receiving message patterns for the n node cluster varied according to OS tasks and the tasks of other users on the nodes in the clusters. This study was not conducted in dedicated cluster environments.

Walker [31] described the load-balancing model that led to the execution times in Figures 2 and 3. The execution results displayed reflect the diversity existing among the different types of computer hardware used in this study. These results also reflect quasi-dedicated computer environments associated with tightly coupled and loosely coupled parallel computer models. The workstation timing results show consistent increases in required CPU time as the training set size increases. The IBM SP2 results display spikes that reflect the impact of other users in a tightly coupled environment, as well as the nondeterministic execution of the load-balancing model.

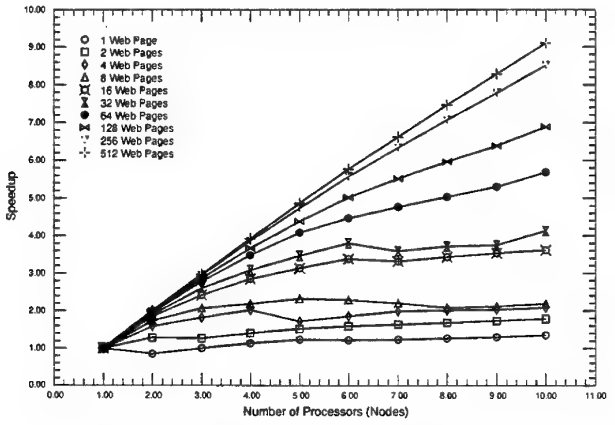


Fig. 4. Speedup for the workstations.

4.2 Network of Workstations

This study showed the limitations that exist on a cluster of quasi-dedicated SUN workstations. The inherent limitations [26] for the parallel implementa-

tions resulted from start-up latencies and limited bandwidth of the Ethernet connections. Employing load balancing for computationally intensive routines led to the most efficient implementations when message passing was reduced. The speedup and efficiency results presented in Figures 4 and 6 were computed using

$$Speed - up = \frac{T_1}{\frac{T}{n_p} + T_{com}} \tag{1}$$

and

$$Efficiency = \frac{Speed - up}{n_p} = \frac{T_1}{T + n_p T_{com}} \tag{2}$$

where T_{com} denotes the communication time and n_p denotes the number of nodes in the sub-cluster.

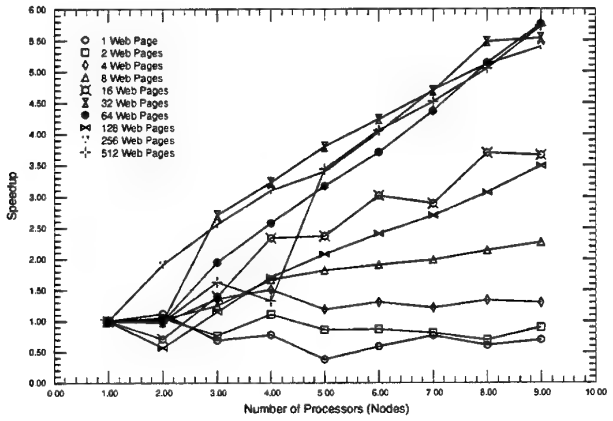


Fig. 5. Speedup for the IBM SP2.

4.3 The IBM SP2

A second study was conducted on the IBM Scalable POWER parallel system 9076 SP2 [23]. This environment supports the MPI language coupled with the SP2's Message Passing Library (MPL). The classification of the programming model supported by this environment is recognized as a distributed memory model, as opposed to a network of workstations. The development of the SP2 resulted from the need for fast communication hardware for parallel data transmissions.

The ideal load-balancing model for the SP2 environment exists when node interaction incorporates the posting of nonblocking send and receive calls. An

efficient implementation of this approach requires that each node post a non-blocking receive and then execute the send. This strategy will allow the send operator to proceed without additional buffering. Likewise, the timing of wait calls should coincide with the availability of the application buffers. Adhering to these implementation techniques may reduce the degree of nondeterministic execution of the load-balancing model by preserving order and reducing the risk of deadlock. The speedup and efficiency results [21] presented in Figures 5 and 7 reflect the nondeterministic execution of the load-balancing model.

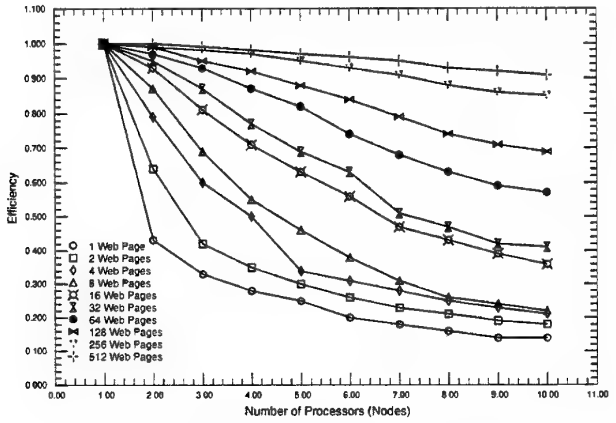


Fig. 6. Efficiency for the workstations.

4.4 Discussion

The communication hardware and page size irregularity affect the load distribution of the Web pages. These effects show in the uneven distribution of pages, as well as the erratic behavior in the execution times in Figures 2 and 3. The nondeterministic execution of the receives by the program manager [19] may be reduced by incorporating the order of receive/send operators for the clients (Web sites) requesting Web pages from the Indexer program manager.

The timing results associated with the network of workstations generated were expected. The predicted output for this environment is also displayed in the speedup and efficiency results. The self-scheduling, load-balancing model [21] indicated that increases in the workload will improve efficiency. The use of 512 Web pages showed that the model will provide an ideal starting point for increasing the workload (addition of the genetic operators) without increasing the number of Web pages. The speedup and efficiency results associated with the IBM SP2 point out the need for a specific load-balancing model.

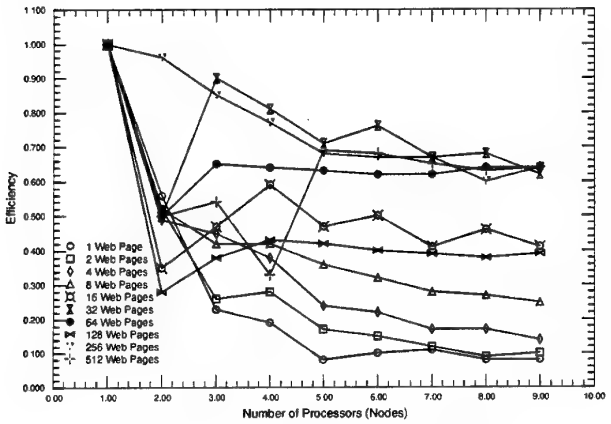


Fig. 7. Efficiency for the IBM SP2.

5 Mechanisms for Expanding the Current Load Balancing Model

5.1 Overview

The social structure associated with *honeybee hierarchy* provides an ordered structure to what can be referred to as the simplest solution to the problem of multiway rendezvous. The initial implemented load-balancing model used a MPI algorithm [19] as the basis. This model followed the approach of a node manager for distributed computing. Similar approaches have been implemented for general distributive computing, as well as for the implementation of parallel GP load-balancing models. The implementation of the load-balancing mechanism for the Pseudo-Search Engine Indexer model follows the theory associated with the implementation of an Event Manager (EM) [2].

The EM concept provides a paradigm for the development and implementation of interface mechanisms associate with the three major components of the Pseudo-Search Engine. The manager interface paradigm will be an extension of the *multiway rendezvous* model [2]. This model provides the following benefits: 1) an extension of the binary rendezvous model where communication involved the synchronization of exactly two nodes, and 2) mechanisms for the synchronous communication between an arbitrary number of asynchronous nodes. These interface components of the Pseudo-Search Engine managers are:

1. Web page indexer manager, M_1
2. Web crawler manager, M_2
3. Web browser interface manager, M_3

Each of these three managers will control its respective load-balancing mechanisms based on its respective functionality.

5.2 Foraging Web Scouts/Crawlers for the Pseudo-Search Engine: A Active Networks Approach Using Genetic Programming

Overview of Web Scouts/Crawlers. The efficiency of Internet applications is being tested by the addition of new applications that compete for the same network resources. Studies associated with network traffic [5],[16] show the need for adaptive congestion control and avoidance at the application level. The side-effects of the current non-adaptive application mechanisms result in self-similarity among network transmissions. The need of efficient Web scouts (probes) for the Pseudo-Search Engine Web crawlers results from the future requirements associated with new applications. The exponential growth of Web documents, the incorporation of multimedia applications with real-time demands, and a steady increase in WWW users will lead to refinements in efficient design and implementation of crawler mechanisms. The competition for bandwidth will reward the adaptive and efficient applications. The incorporation of active networks (ANs) methodologies [27],[28] can enhance the development and incorporation of the biological model associated with the Pseudo-Search Engine.

Aspects of the foraging mechanisms used by the bee colony provide a basis for scout/crawler mechanisms to be used for congestion control and data transmission [13]. The factors that influence the amount of foraging are temperature, weather, and day length. The weather affects the availability of pollen and nectar. The temperature coupled with the time of day determines the quantity of pollen and/or nectar. The attractiveness of particular crops are rated based on several criteria. Similar mechanisms are needed to determine the routing tables for retrieving Web pages from distributed computer networks that span the Internet and provide a diversity of resources.

Overview of Active Networks. Active networks research provides insight into the software needed to support GP communicating agents being developed to retrieved WWW documents from the diverse set of Web sites. ANs enable the retrieval of state information from routers that support the infrastructure of the Internet by embedding active capsules (components) [11] within each packet transmitted via the Internet. The active capsules are executed on the routers as the packets traverse the Internet starting at a source (host) and possibly terminating at the destination (host).

Active networks was developed to facilitate efficient network communication [35] by incorporating active capsules in the packets that are executed and routed by the switches that support the Internet's infrastructure. The ANs model was designed to minimize additional computational overhead at the router level needed to activate the capsule, but the overhead increases based on the complexity of the transmitted active capsule component. Additional complexity can be added to the basic AN model through the enhancement of the execution environments (EEs) [3] which result in virtual EEs. This area of research provides execution environments which can be used to program routers to capture state information associated with LANs and/or WANs, which in turn can be incorpo-

rated into a methodology for creating scouts/crawlers needed for the retrieval of Web pages for the Pseudo-Search Engine [30].

5.3 Proposed Web Scout and Crawler Mechanisms

The Web crawlers required to adequately retrieve the growing number of Web pages will require some form of adaptive methodology as each Web scout (probe) searches for efficient paths (routes) to an adequate source of information (Web documents) to build Internet Service Provider (ISP) router tables for the crawler mechanisms. The initial step of this proposed methodology is to send out scouts to all ISP providers in a manner similar to reliable flooding [20]. The purpose of collecting timing and path information to and from the ISP providers reflects the need to find efficient routes to the portal associated with the hosts of information reflected in its hierarchy structure of sub-hosts. Each provider is viewed as a gateway into the information associated with its sub-host Web page directory structure. This methodology has the ability to discover new ISPs, as well as new sub-hosts providing services to new and existing Web clients. The end effect is the faster discovery of new Web pages.

5.4 Strategies for Communicating Agents Using Genetic Programming

Iba et al. [12] presented studies of communicating agents that reflect the need to evaluate techniques for developing cooperating strategies. One application of this methodology is the Predator-Prey pursuit problem – a test bed in Distributed Artificial Intelligence (DAI) research – that measured the impact of limited ability and partial information for agents pursuing/seeking the same goal independently, instead of relying on cooperation to solve a discrete set of subproblems. The metrics associated with this aspect of GP research included: 1) applicability of GP to multi-agent test beds, 2) observing the robustness (brittleness) of cooperative behavior, and 3) examining the effectiveness of communication among multiple agents. This co-evolutionary strategy provides a methodology for the comprehensive assessment of the impact of robustness (brittleness) of cooperative behavior and its effectiveness among communicating agents. The robustness of a GP program was defined [12] as the ability of agents to cope with noisy or unknown situations (unknown test data) within a GP application when communication among multiple agents was due to effective work partitioning. New and potentially improved behavior patterns were found to evolve through the use of a fitness measure associated with a co-evolutionary strategy. The panoply (multiplicity) of relationships among the communicating agents include:

- Agents requesting data from other agents (Communicating Agents)
- Agents negotiating their movements with other agents (Negotiating Agents)
- Agents controlling other agents (Controlling Agents)

6 Conclusion

The current Pseudo-Search Engine Indexer, capable of organizing limited subsets of Web documents, provides a foundation for the first beehive simulators. Adaptation of the honeybee model for the refinement of the Pseudo-Search Engine establishes order in the inherent interactions between the indexer, crawler and browser mechanisms by including the social (hierarchical) structure and simulated behavior of the honeybee model. The simulation of behavior will engender mechanisms that are controlled and coordinated in their various levels of complexity.

7 Acknowledgments

The author wishes to thank Walter Karplus, Zhen-Su She, and Peter Reiher for their direction and suggestions, and Elias Houstis, Ahmed K. Elmagarmid, Apostolos Hadjidimos, and Ann Catlin for support, encouragement, and access to the computer systems at Purdue University. Special thanks to the 1999 UCLA SMARTS students (London Wright-Pegs and Samir Asodia) and Martha Lovette for assistance with the chromosome structures depicted in Figure 1. Plots were generated using *PSplot* written by Paul Dulaney (Raytheon Corporation).

Support for this work came from the Raytheon Fellowship Program and Honeybee Technologies. Implementation results associated with the network of workstations originated on computers located in the Department of Computer Sciences, Purdue University, West Lafayette, IN. The implementation results associated with the IBM SP2 were generated on a computer cluster located at the University of California, Los Angeles.

References

1. Abramson, M.Z., Hunter, L.: Classification using Cultural Co-evolution and Genetic Programming. In: Koza, J.R., Goldberg, D.E., Fogel, D.B., Riolo, R.L. (eds.): Proc. of the 1996 Genetic Programming Conf. MIT Press, Cambridge, MA (1996) 249–254.
2. Bagrodia, R.: Process Synchronization: Design and Performance Evaluation of Distributed Algorithms. IEEE Transactions on Software Engineering **15 no. 9** (1989) 1053–1064.
3. Braden, B., Cerpa, A., Faber, T., Lindell, B., Phillips, G., Kann, J.: The ASP EE: An Active Execution Environment for Network Control Protocols. Technical Report, Information Sciences Institute, University of Southern California, Marina del Rey, CA (1999).
4. Chapman, C.D., Jakiela, M.J.: Genetic Algorithm-Based Structural Topology Design with Compliance and Topology Simplification Considerations. J. of Mech. Design **118** (1996) 89–98.
5. Crovella, M.E., Bestavros, A.: Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes. IEEE/ACM Transactions on Networking (1997) 1-25.

6. Dracopoulos, D.C., Kent, S.: Bulk Synchronous Parallelisation of Genetic Programming. In: Wasniewski, J., Dongarra, J., Madsen, K., Olesen, D. (eds.): PARA'9: Proc. of the 3rd Intl. Workshop on Applied Parallel Computing, Industrial Computation and Optimization. Springer-Verlag, Berlin, Germany (1996) 216–226.
7. Duda, J.W., Jakiela, M.J.: Generation and Classification of Structural Topologies with Genetic Algorithm Speciation. Journal of Mechanical Design **119** (1997) 127–131.
8. Franke, H., Hochschild, P., Pattnaik, P., Snir, M.: An Efficient Implementation of MPI. In: Proc. of Conf on Prog. Environments for Massively Parallel Distributed Systems. (1994) 219–229.
9. Free, J.B.: The Social Organization of Honeybees (Studies in Biology no. 81). The Camelot Press Ltd, Southampton (1970).
10. Gouet, P., Diprose, J.M., Grimes, J.M., Malby, R., Burroughs, J.N., Zientara, S., Stuart, D.I., Mertens, P.P.C.: The Highly Ordered Double-Stranded RNA Genome of Bluetongue Virus Revealed by Crystallography. Cell **97** (1999) 481–490.
11. Horta, E.L., Kofuji, S.T.: Using Reconfigurable Logic to Implement an Active Network. In: Shin, S.Y. (ed.): CATA 2000: Proc. of the 15th Intl. Conf. on Computers and their Applications. ISCA Press, Cary, NC (2000) 37–41.
12. Iba, H., Nozoe, T., Ueda, K.: Evolving Communicating Agents based on Genetic Programming. In: ICEC '97: Proc. of the 1997 IEEE Intl. Conf. on Evolutionary Computation. IEEE Press, New York (1997) 297–302.
13. Information Sciences Institute: Transmission Control Protocol (TCP). Technical Report RFC: 793, University of Southern California, Marina del Rey, CA (1981).
14. Koza, J.R.: Survey of Genetic Algorithms and Genetic Programming. In: Proc. of WESCON '95. IEEE Press, New York (1995) 589–594.
15. Koza, J.R., Andre, D.: Parallel Genetic Programming on a Network of Transputers. Technical Report STAN-CS-TR-95-1542. Stanford University, Department of Computer Science, Palo Alto (1995).
16. Leland W.E., Taqqu M.S., Willinger W., Wilson, D.V.: On the Self-Similar Nature of Ethernet Traffic. In: Proc. of ACM SIGComm '93 ACM Press (1993) 1-11.
17. Marenbach, P., Bettenhausen, K.D., Freyer, S., U., Rettenmaier, H.: Data-Driven Structured Modeling of a Biotechnological Fed-Batch Fermentation by Means of Genetic Programming. J. of Systems and Control Engineering **211 no. I5** (1997) 325–332.
18. Oussaidène, M., Chopard, B., Pictet, O.V., Tomassini, M.: Parallel Genetic Programming and Its Application to Trading Model Induction. Parallel Computing **23 no. 8** (1997) 1183–1198.
19. Pacheco, P.S.: Parallel Programming with MPI. Morgan Kaufman Publishers, Inc., San Francisco, (1997).
20. Peterson, L.L., Davie, B.S.: Computer Networks: A Systems Approach. Morgan Kaufmann Pbulishers, Inc., San Francisco (1996).
21. Quinn, M.J.: Designing Efficient Algorithms for Parallel Computers. McGraw-Hill, New York (1987).
22. Sherrah, J., Bogner, R.E., Bouzerdoum, B.: Automatic Selection of Features for Classification using Genetic Programming. In:Narasimhan, V.L. Jain, L.C. (eds.): Proc. of the 1996 Australian New Zealand Conf. on Intelligent Information Systems. IEEE Press, New York (1996) 284–287.
23. Snir, M., Hochschild, P., Frye, D.D., Gildea, K.J.: The communication software and parallel environment of the IBM SP2. IBM Systems Journal **34 no. 9** (1995) 205-221.

24. Stoffel, K., Spector, L.: High-Performance, Parallel, Stack-Based Genetic Programming. In: Koza, J.R., Goldberg, D.E., Fogel, D.B., Riolo, R.L. (eds.): Proc. of the 1996 Genetic Programming Conf. MIT Press, Cambridge, MA (1996) 224–229.
25. Tanese, R.: Parallel Genetic Algorithm for a Hypercube. In: Grefenstette, J.J. (ed.): Proc. of the 2nd Intl. Conf. on Genetic Algorithms. Lawrence Erlbaum Associates, Hilsdale, NJ (1987) 177–183.
26. Tatsumi, M., Hanebutte, U.R.: Study of Parallel Efficiency in Message Passing Environments. In: Tentner, A. (ed.): Proc. of the 1996 SCS Simulation Multiconference. SCS Press, San Diego, CA (1996) 193–198.
27. Tennenhouse, D.L., Smith, J.M., Sincoskie, W.D., Wetherall, D.J., Minden, G.J.: A Survey of Active Network Research. IEEE Communications Magazine **35 no. 1** (1997) 80-86.
28. Tennenhouse, D.L., Wetherall, D.J.: Towards an Active Network Architecture. ACM Computer Communications Review **26 no. 2** (1996).
29. von Frisch, K.: Bees: Their Vision, Chemical Senses, and Languages. Cornell University Press, Ithaca, New York (1964).
30. Walker, R.L.: Assessment of the Web using Genetic Programming. In: Banshaf, W., Daida, J., Eiben, A.E., Garzon, M.H., Honavar, V., Jakiela, M., Smith,R.E. (eds.): GECCO-99: Proc. of the Genetic and Evolutionary Computation Conf. Morgan Kaufman Publishers, Inc., San Francisco (1999) 1750–1755.
31. Walker, R.L.: Development of an Indexer Simulator for a Parallel Pseudo-Search Engine. In: ASTC 2000: Proc. of the 2000 Advanced Simulation Technologies Conf. SCS Press, San Diego, CA (April 2000) To Appear.
32. Walker, R.L.: Dynamic Load Balancing Model: Preliminary Assessment of a Biological Model for a Pseudo-Search Engine. In: Biologically Inspired Solutions to Parallel Processing Problems (BioSP3). Lecture Notes in Computer Science. Springer-Veglag, Berlin Heidelberg New York (2000) To Appear.
33. Walker, R.L.: Implementation Issues for a Parallel Pseudo-Search Engine Indexer using MPI and Genetic Programming. In: Ingber, M., Power, H., Brebbia, C.A. (eds.): Applications of High-Performance Computers in Engineering VI. WIT Press, Ashurst, Southampton, UK (2000) 71–80.
34. Walker, R.L., Ivory, M.Y., Asodia, S., Wright-Pegs, L.: Preliminary Study of Search Engine Indexing and Update Mechanisms: Usability Implications. In: Shin, S.Y. (ed.): CATA 2000: Proc. of the 15th Intl. Conf. on Computers and their Applications. ISCA Press, Cary, NC (2000) 383–388.
35. Wetherall, D.J.: Developing Network Protocols with the ANTS Toolkit. Design Review (1997).
36. Willis, M.J., Hiden, H.G., Marenbach, P., McKay, B. Montague, G.A.: Genetic Programming: An Introduction and Survey of Applications. In: Proc. of the 2nd Int. Conf. on Genetic Algorithms in Engineering Systems: Innovations and Applications. IEE Press, London (1997) 314–319.

A novel collective communication scheme on packet-switched 2D-mesh interconnection

MinHwan Ok and Myong-Soon Park

Internet Computing Laboratory
Department of Computer Science and Engineering, Korea University
Sungbuk-ku, Seoul 136-701, Korea
{mhok, myongsp}@ilab5.korea.ac.kr

Abstract. Recently, cluster computing which employs many cheap node machines is going to replace expensive supercomputers. However, there exist only a few of enhanced communication schemes for cheap packet-switches, especially in case of collective communication. We devised a new collective communication scheme from original dimension-order routing. The proposed scheme mainly aims at non-uniform traffic situation by communication locality that causes longer communication delays than that in uniform-traffic situation. By addition of 'flow bit' in each packet, packets can traverse alternating their directions at hop by hop. The new scheme is devised for 2D mesh and enhanced the original X-Y routing.

1. Introduction

There are broadly two categories of parallel computers with respect to the types of their memories. These physical models are distinguished by having a shared common memory among each processor or unshared distributed memory in each processor. In both, interconnection between processors or memories considerably affects whole processing capacity of the parallel computer.

Interconnection networks are classified into two major classes primarily based on interconnection topology. If all adjacent nodes are connected to each other with direct link, the interconnection is called direct network. Otherwise, the network is called indirect network. The direct network or point-to-point network consists of a set of nodes, each one being directly connected to a subset of other nodes in the network. Each node is a programmable computer with its own processor, local

memory, and other supporting devices. These nodes may have different functional capabilities. As the number of nodes in the system increases, the total communication bandwidth, memory bandwidth, and processing capability of the system also increase. Thus, direct networks have been popular interconnection architecture for constructing large-scale parallel computers.

When the network traffic is non-uniform, there is the probability of a phenomenon which one node or a subset with a few nodes accounts for a disproportionately large portion of the total network traffic. This phenomenon is inherent from parallelizing of the entire work. It is crucial if the phenomenon arises often from short communications occurring frequently in fine-grained parallel processing and incurs many short delays that eventually make entire processing slow. The more frequent messaging, the higher probability of the phenomenon.

Mesh is one of popular interconnection networks among many direct networks. Till now, mesh interconnection necessarily facilitated wormhole switching. Recently, cluster computing which employs many cheap node machines is going to replace expensive supercomputers. However, there exist only a few of enhanced communication scheme for cheap packet-switches, especially in case of collective communication. We devised a new collective communication scheme from original dimension-order routing.

This paper comprises 5 sections: the introduction, prior studies on non-uniform traffic, a novel routing scheme of mesh, comparison with simulation and summary of the novel scheme.

2. Prior Studies on Non-uniform Traffic

One major cause of non-uniform traffic is not-even data-access. As much traffic was necessitated around any particular location, this phenomenon is called 'hot spot'. Another major cause is group communication among processors including such as global synchronization operation.

Contention caused by accesses to the hot spot is notorious for degrading performance of a parallel algorithm [1]. The approach incorporates certain hardware in the interconnection network to trap and combine access requests for hot spot relief. However, the cost overhead due to added hardware posed a major concern. A less costly hardware combining techniques was introduced in [4]. Combining messages reduces communication traffic and decreases the average amount of buffer space used, what leads to a lower average network delay time. The delay time experienced by a message traversing a buffered multistage network could be enhanced with message combining under hot spot traffic, also in hardware, in [2] and [3].

Like non-uniform traffic, blocked message also disturbs fluent network traffic. Deadlock-free routing and adaptive routing were proposed to prevent low network utilization from slowdown in propagation of messages [5][6]. Deadlock or saturation of the message routes has the potential to degrade system performance in terms of lower throughput and higher latency similar to that of hot spot.

Recent approaches to alleviate the effect of non-uniform traffic are shown in variety.

Lee and Chen have proposed an allocation method of hot spots on mesh [7]. They divided hot spots into two classes, one to be uniform and the other to be non-uniform. They have presented how to minimize hot spot access time in case of uniform hot spot. However, They let the case of non-uniform hot spot opened. Wang et al. has used extra paths to alleviate hot spot problem on multipath MIN [8]. Their approach is to force all hot spot messages to choose some predefined paths and non-hot spot messages not to choose the paths that involve any interchange boxes in the saturated area or to have only limited overlap. In the approach, additional links are necessary thus additional hardware cost is imposed. Fu and Tzeng have proposed a scheme to keep synchronization traffic low and avoid hot spot contention for shared memory systems [9]. Their scheme constructs a circular list of processors waiting for the critical section by dispersing access to the lock. The scheme is also founded on MIN. Basak and Panda have used multiple consumption channels than single one for each processor on wormhole-routed k-ary n-cube [10]. Their approach is to analyze various factors of interconnection network with message consumption, and derive the minimum number of required consumption channels for alleviating consumption bottleneck. In the approach, additional channel per processor is necessary thus additional hardware cost is also imposed.

3. A Novel Routing Scheme of Mesh

Many approaches have been proposed for efficient collective communication with wormhole switching [11]. Among the approaches, there are mainly two methodologies of either tree-based or path-based. Tree-based and path-based routings require message partitioning and contention-free condition, to reduce blocking time, respectively. They use wormhole-routers for interconnection and the routers are rather expensive. If routers do not support special functions such as wormhole switching, the approaches cannot be exploited. However, suggested routing schemes for collective communication with little special functionality are relatively rare.

From the programmer's view, the unit of information exchange is a message. For efficient and fair use of network resources, a message is divided into packets prior to transmission. Message passing is divided into two classes, one for point-to-point communication and the other for collective communication. When packet switching is applied, collective communications essentially incur non-uniform traffic situation. Where the non-uniform traffic situation arise, the center of the situation, becomes 'hot spot.' As analyzed in [1], each node around 'hot spot' gets its buffer full, incurring concentrated messages suffer high latency due to wait times in each node's buffer. Moreover, this phenomenon has influence upon messages that pass by around the location. Thus, it is necessary that collective communications should be concerned with separate scheme from that of point-to-point communication to alleviate the effect of non-uniform traffic.

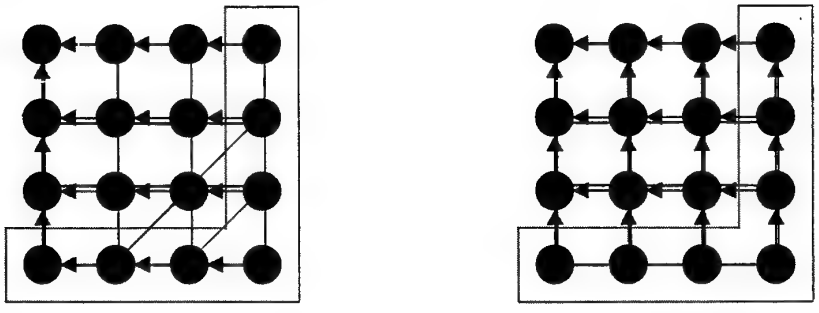
There are many routing schemes according to interconnection topologies. Among the topologies, mesh is one of most popular topologies. In this paper, we consider 2D mesh of 7x7 size. In this interconnection, all the boundary nodes are to send their packets to the one center node. Deterministic dimension-order routing, X-Y routing, is the routing scheme and packet switching is applied.

To simplify the analysism we assume that;

< *Assumption* >

1. Each node has 4 links of which are directing to north, south, west, and east, respectively.
2. A packet transfers by one hop for one unit time.
3. A link is also capable of a packet transmission for one unit time.
4. The packet length is fixed and suitable for the transmission in one unit time.

Let's see only east-southern part of the whole of nodes. The other three parts are the reflections and show the same characteristics.



{A} X-Y Routing {B} The New Routing

Fig. 1. Collective communications in east-southern part of 7x7 mesh

In figure {1-A}, 4 packets suffer conflicts along the leftmost vertical path. Nodes at the same diagonals transmit packets that would conflict at nodes close to the center node between the conflicting packets' Y-dimensional displacements along the leftmost vertical path. For there is one link for each direction and only one packet transfers along that link for one unit time, packets remained are accumulated at the vertical nodes.

When the number of used links are investigated, there are many idle link during packet transmissions. Only one of conflicting packets advances and the other packets are to be remained in buffer waiting for its order in queue. If the idle link can be used for packet transmission, the number of conflicts can be reduced. The conflict occurs when both the horizontal link and the vertical link receive packets with the same next direction. Thus, some conditions are set here;

< *Condition* >

1. When a packet comes in the node through any horizontal link, it should go out the node through any vertical link.
2. When a packet comes in the node through any vertical link, it should go out the node through any horizontal link.

And a theorem;

< *Theorem* >

For the times the above conditions are satisfied, a node of 2D mesh is able to receive packets with 2 adjacent links at the same time and it is able to send them simultaneously with the other 2 adjacent links at the next time if the node's buffer has no packet to send prior to them.

By the theorem, we can set the packet's next directions in advance to make use of the links at higher utilization. The theorem is to avoid conflicts of next directions.

From the idea, a new routing scheme permits packet transmission at every node that alternates its direction heading for its destination. The new scheme is described in <*Algorithm*>.

In figure {1-B}, new routing scheme makes use of vertical idle links from the figure {1-A}. To keep high the utilization of links, as previously mentioned, both the horizontal link and the vertical link should receive packets at same time as little as possible in this "one packet transmission for one unit time" network model. With the same preset 'flow bit' of each packet, every packet flows with the same direction, until no conflict occurs. The same direction of every packet ensures them against conflict. Where a conflict occurred, the other packet is to be remained in buffer. As all the packets approach either X-axis or Y-axis in the first halves of their routes, when compared with the original X-Y routing, there are two advantages in the new

scheme. One is lower possibility of conflicts and the other is higher utilization of available links.

I. Initialization
Let every packet has its flow bit.
Every flow bit is set 0.
Each flow bit is set before the first transmission.

II. Routing
For the case of flow bit;
0 : Advance along Y-dimension. Change the flow bit to 1.
If Y coordinates, advance along X-dimension.
1 : Advance along X-dimension. Change the flow bit to 0.
If X coordinates, advance along Y-dimension.

< *Algorithm* > A novel routing algorithm for 2-D mesh

4. Simulation

An important metric to evaluate a network throughput by a modified routing scheme is communication latency, which is the sum of three values: start-up time, transmission latency, and blocking time. The start-up time is the time required for message framing/unframing, memory/buffer copying, validation, and so on. The transmission latency is the time elapsed after the head of a message has been injected into network at the source node until the tail of the message is extracted from the network at the destination node. The blocking time includes all possible delays encountered during the lifetime of a message. For given a source and destination node, the start-up time and the transmission latency are static values. In this paper, we define the parameters as follows for simplicity and ease of comparison:

Transmission latency: latency by a packet to traverse its route until arrival at the destination
Blocking time: elapsed time in a node buffer due to busy communication link in use

The start-up time is not considered and it is relatively very small and fixed time. Thus communication latency C of one packet is defined as follows;

$$C = T + B, \tag{1}$$

where T is the transmission latency and B is the blocking time. Since we assumed that one packet is transmitted along one link for one unit time, T is same as the hop count. The blocking time is same as the number of unit times for wait time in the buffer. Therefore, blocking time of X-Y routing is 4 unit times while that of new routing scheme is 0 unit times in case of <Fig 1>.

Let us consider the case of all-to-one collective communication in <Fig.2>. All the nodes are to send their packets to the one center node, one packet per one node and we see only east-southern part of 7x7 mesh.

With original X-Y routing, every packet goes through X-path in the first half of the transmission and through Y-path in the second half of the transmission. In figure {2-A}, 3 of 4x4 packets should go through Link 2 and the others should go through Link 1. If only this east-southern part is considered, that results in at least 12 unit times as communication latency since 4x3 packets should go through Link 1. With new routing, every packet goes through X-direction and Y-direction alternately in the transmission. In figure {2-B}, 9 of 4x4 packets go through Link 2 and the others go through Link 1. Again if only this east-southern part is considered, that results in at least 9 unit times as communication latency since 1+2+3+3 packets should go through Link 1.

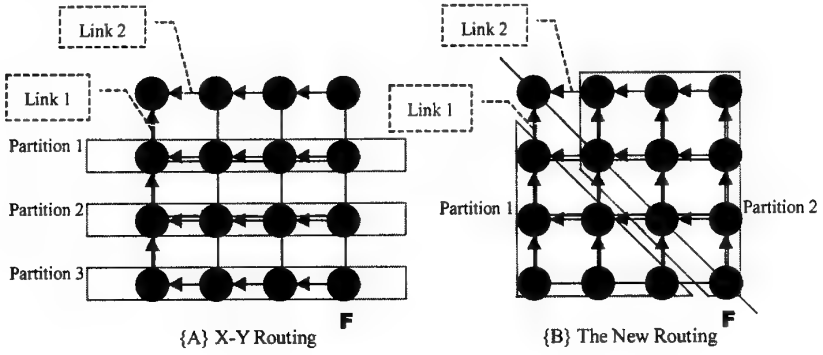


Fig. 2. All-to-one collective communications with X-Y routing and with the new routing

For ease of comparison, suppose each node has two queues, one for packets that come in through X-dimensional link, say X-queue, and the other for packets that come in through Y-dimensional link, say Y-queue. We put packets of X-queue in higher priority so packets of Y-queue cannot advance until X-queue empties. Then with X-Y routing, packets from nodes of Partition 1, those from Partition 2, and those from Partition 3 would arrive at the destination in sequence of Partitions. With the

new routing, packets from Partition 1 and Partition 2 would arrive at the destination in parallel.

As the final packet arrived at the destination determines the collective communication time, the packet from 'node F' finishes the collective communication in either routing. In figure {2-A}, packets from Partition (m) cannot advance along Y-dimension until packets from Partition (m-1) do. For nxn mesh, the communication latency of the final packet in all-to-one collective communication with X-Y routing;

$$C_{XY} = nm, \tag{2}$$

where m is the number of Partitions. For nxn mesh, the communication latency of the final packet in all-to-one collective communication with new routing;

$$C_{Novel} = nm - \sum_{k=1}^{m}(2k-1) = nm - m^2, \tag{3}$$

where m is the number of Segments in a Partition and a Segment is each row in the Partition. For nxn mesh,

$$n = 2m + 1. \tag{4}$$

Thus communication latencies of both routings are;

$$C_{XY} = 2m^2 + m, \tag{5}$$

and

$$C_{Novel} = m^2 + m. \tag{6}$$

The new routing scheme is simulated under previously stated <*Assumption*>. Topology is 16x16 2D-mesh and packet switching is applied. Distribution of message locations is random uniform, and the collective messages are to come to one center node. <Fig. 3> shows comparison of the original X-Y routing and the new scheme. In addition to collective messages, there also exist another point-to-point messages. It is to examine whether the new routing scheme has aided packets pass by around the center node by reducing conflicts. The total of messages is 128 for every case. The number of collective messages is same with the number of nodes in the collective communication group, and the number of point-to-point messages is the remainder. In the new scheme, collective communication routing is the new devised one and point-to-point routing is the original X-Y one.

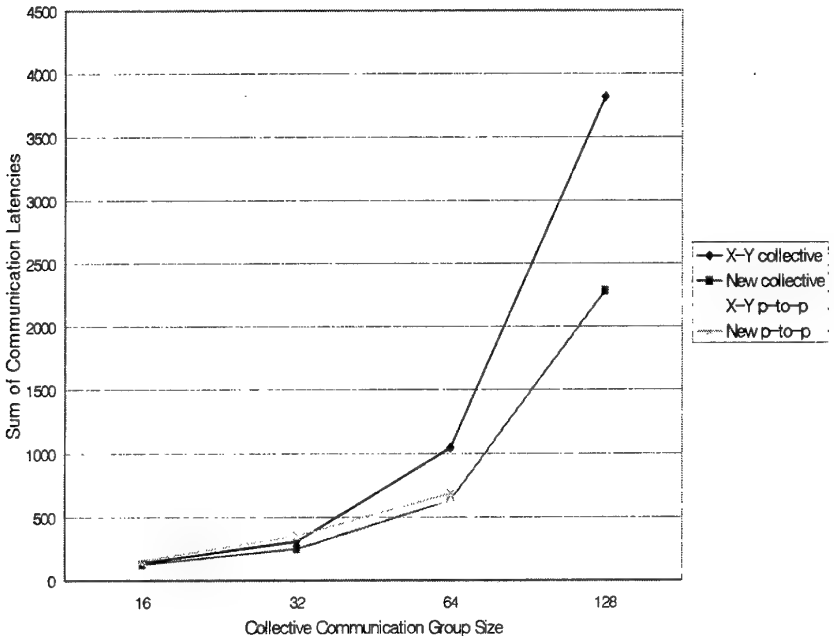


Fig. 3. Comparison of sum of communication latencies on 16x16 mesh

In <Fig. 3>, we can see the new scheme perform better than the original X-Y as the number of collective messages increases. It is due to messages that form two rows on either axis heading for the destination. <Fig. 4> shows comparison of the largest latencies in both collective communications.

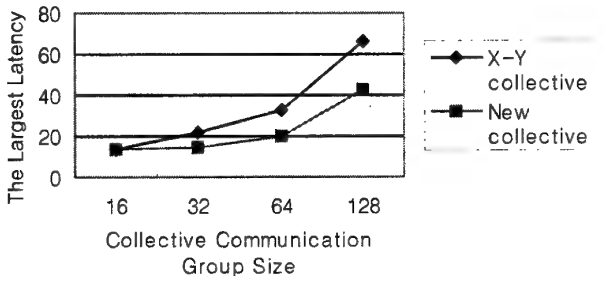


Fig. 4. Comparison of the largest communication latency with either routing

5. Summary

The proposed scheme mainly aims at non-uniform traffic situation by communication locality that causes longer communication delays than that in uniform-traffic situation. Collective communication or operation including broadcast, scatter, gather, and reduce essentially incur non-uniform traffic situation since at least acknowledgements are required in any communication. By addition of 'flow bit' in each packet, packets can traverse making use of idle links. The new scheme is devised for 2D mesh and enhanced the original X-Y routing. Compared with the original X-Y routing, there are two advantages in the new scheme, one is lower possibility of conflicts and the other is higher utilization of available links. Simulation results that when traversing packets' lengths are all the same, reducing the wait time in buffer by making use of available links shows a consequence of lower communication latency. This new scheme is devised for cheap packet-switches, and expected to contribute toward constructing cluster's interconnection for more efficient collective communication with slightly increased implementation cost.

References

1. G. F. Pfister and V. A. Norton, "Hot spot contention and combining in multistage interconnection networks," *IEEE Trans. Computers*, vol. 34, pp. 943-948, Oct. 1985
2. G. Lee, "A performance bound of multistage networks," *IEEE Trans. Computers*, vol. 38, pp. 1387-1395, Oct. 1989
3. B. –C. Kang, G. Lee, and R. Kain, "Performance of multistage combining networks," *Proc. 1991 Int. Conf. Parallel Processing*, pp.550-557, Aug. 1991
4. N. –F. Tzeng, "A cost-effective combining structure for large-scale shared-memory multiprocessors," *IEEE Trans. Computers*, vol. 41, pp. 1420-1429, Nov. 1992
5. J. Duato, "A new theory of deadlock-free adaptive routing in wormhole networks," *IEEE Trans. Parallel and Distributed Systems*, vol. 4, pp. 1320-1331, 1993
6. Y. M. Boura and C. R. Das, "Efficient fully adaptive wormhole routing in n-dimensional meshes," *Proc. Int'l Conf. Distributed Computing Systems*, pp. 589-596, 1994
7. S. –Y. Lee, C. –M. Chen, "Optimal hot spot allocation on meshes for large-scale data-parallel algorithms," *IEEE Trans. Parallel and Distributed Systems*, vol. 6, pp. 788-802, Aug. 1995
8. M. –C. Wang, H. J. Siegel, M. A. Nichols, S. Abraham, "Using a multipath network for reducing the effect of hot spots," *IEEE Trans. Parallel and Distributed Systems*, vol. 6, pp. 252-268, Mar. 1995

9. S. S. Fu, N. –F. Tzeng, "A circular list-based mutual exclusion scheme for large shared-memory multiprocessors," *IEEE Trans. Parallel and Distributed Systems*, vol. 8, pp. 628-639, Jun. 1997
10. D. Basak, D. K. Panda, "Alleviating consumption channel bottleneck in wormhole-routed k-ary n-cube systems," *IEEE Trans. Parallel and Distributed Systems*, vol. 9, pp.481-496, May 1998
11. P. K. McKinley, D. F. Robinson, "Collective communication in wormhole-routed massively parallel computers," *IEEE Computer*, vol. 28, pp. 39-50, Dec. 1995

Enhancing Parallel Multimedia Servers through New Hierarchical Disk Scheduling Algorithms

Javier Fernández, Félix García, Jesús Carretero

Dto. de Informática, Universidad Carlos III de Madrid,
Avda. Universidad 30, Leganes , 28991, Madrid, Spain.
{jfernand, fgarcia, jcarrete}@arcos.inf.uc3m.es

Abstract. [1] An integrated storage platform for open systems should be able of meeting the requirements of deterministic applications, multimedia systems, and traditional best-effort applications. It should also provide a disk scheduling mechanism fitting all those types of applications. In this paper, we propose a three-level hierarchical disk scheduling scheme, which has three main components: *metascheduler*, single server scheduler, and disk scheduler. The *metascheduler* provides scheduling mechanisms for a parallel disk system or a set of parallel servers. The server level is divided in three main queues: deterministic, statistic and best-effort requests. Each server may have its own scheduling algorithm. The lower level, disk driver, chooses the ready streams using its own scheduling criteria. Those systems have been implemented and tested, and the performance evaluations demonstrate that our scheduling architecture is adequate for handling stream sets with different timing and bandwidth requirements.

1 Introduction

Over the last years, there has been a great interest on the scheduling of I/O devices, usually disks, in computer systems [16, 9]. However, the requirements and the platforms for both multimedia and general systems seemed to be so different [7], that people developed specialized systems. Thus, disk scheduling algorithms for general purpose systems tried to reduce the access time, while multimedia systems tended to satisfy the real-time constraints for *cyclical* streams, even loosing performance. With the multimedia applications increasing, some authors [15] have proposed the design of a new kind of system, named *integrated*, that include facilities to support heterogeneous multimedia and general purpose information. In an *integrated* system, the user may request the start of a new I/O request (stream) during run-time. The system must determine, following some admission criteria, whether the stream is schedulable to admit or reject

[1] This work has been supported in part by the NSF Award CCR-9357840, the contract DABT63-94-C-0049 from DARPA and by the Spanish CICYT under the project TIC97-0955

the stream. The main problem with most of those systems is that they do not provide schedulability guarantees for deterministic applications [6].

In this paper we describe a hierarchical three-level disk scheduling scheme, which is actually used in *MiPFS*, a multimedia integrated parallel file system [3]. The scheduler has three main components: *metascheduler*, single server scheduler, and disk scheduler. The *metascheduler* provides a scheduling mechanisms for parallel disk system or a set of parallel servers. The server level of the architecture, is divided in three main queues: deterministic, statistic and best-effort requests. Each server may have its own scheduling algorithm. The lower level, disk driver, chooses the ready streams using its own scheduling criteria. We also propose an adaptive admission control algorithm relying on worst and average values of disk server utilization. Only streams satisfying the admission algorithm criteria [2] are accepted for further processing by the disk server. Section 2 presents some related works. Section 3 describes the multi-level architecture of our scheduling scheme, including the extension to a parallel disk system. Section 4 presents some performance evaluations of our scheduling architecture, which were first simulated, and then implemented and tested. Finally, section 5 shows some concluding remarks and future work.

2 Related Work

There are well known techniques to schedule deterministic real-time tasks [14, 13]. However, most of them are oriented to fixed priority scheduling, or at most to dynamic priority scheduling in very specific situations. However, in I/O devices, the timing requirements of many streams are not known in advance, thus a global scheduling analysis can not be done [12]. The priority of the streams must be dynamically computed taking into account parameters, such as time, disk geometry, and quality of service granted to the application. Several algorithms have been proposed to satisfy this timing constraints in multimedia systems: EDF gives the highest priority to I/O streams with the nearest deadline [19], SCAN-EDF order the I/O streams by deadline and the stream with the same deadline by ascending order [16], and SCAN-RT orders the streams using an ascending order but taking into consideration the deadlines [11, 5]. However, none of them addresses the problem of integrated environments, where multiple types of streams must be scheduled. An integrated scheduling scheme should try to reduce the answer time to best-effort requests, but it should also provide a time guided disk scheduler giving more priority to real-time stream with the nearest deadline. Because most of the disk scheduling schemes proposed up to now can hardly make a good trade-off between the former aspects, some multi-level hierarchical scheduling architectures have been proposed for deterministic tasks in an open environment [6], and for integrated multimedia systems [15]. In essence, those schemes create a slower virtual device for each stream or for each class of streams.

We have implemented our scheduling architecture, shown in the next section, on a multiprocessor running LINUX [1, 18]. We have chosen LINUX for several

reasons: it is free, it is updated, and it can include new modules into the OS without modifying the kernel continuously. The last feature is very useful to test different scheduling policies easily.

3 Parallel Disk Scheduler Architecture

Figure 1 shows the scheduling architecture proposed for *MiPFS*. First, the architecture for a single server is described. Then, it is scaled up to show the parallel scheduler architecture (*metascheduler*). Both components are described below.

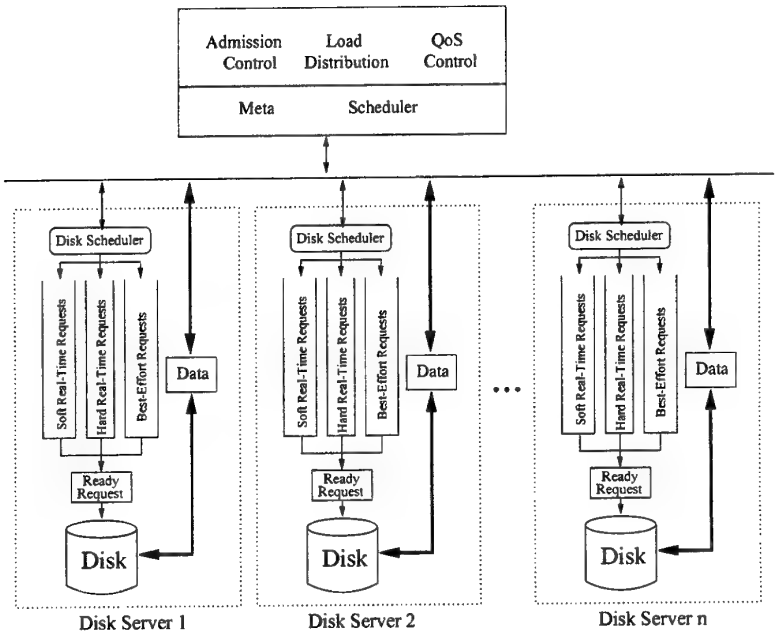


Fig. 1. Parallel Disk Scheduling System Architecture

3.1 Single Server Architecture

Each server scheduler consists of two levels. An upper level with three stream server schedulers: deterministic, statistic, and best-effort streams. A lower level including a disk driver scheduler D, a ready queue R, and a disk. Thus, each scheduling decision involves two steps. First, each server scheduler receives streams, and, based on its particular scheduling algorithm, inserts them into corresponding place in its queue. Second, when the disk is free, D chooses one stream

among the upper level queues, using its own scheduling algorithm, and put it into R. Statistic real-time streams allow a certain percentage of deadline misses, as far as the quality of service (QoS) of the client could be met, the service is deemed OK. Deterministic real-time streams do not allow any misses. In our prototype, the disk scheduling algorithm used at the server's level are: *EDF* for deterministic, *SCAN-EDF* for statistic, and *SCAN* for best-effort streams. The disk driver scheduler D chooses the stream to be executed next based on the dynamic priorities previously computed. In our hierarchical system, each server queue has a different priority that is used by D as a criteria to choose ready jobs. However, using only this priority criteria will be unfair for best-effort and statistic applications, leading to a high percentage of deadline misses. streams without a deadline.

To insert the streams into the servers queues, two major parameters are used in our scheduling scheme: deadline and service time. The service time is totally application dependent because it depends on the track number, while the deadline of a stream may be modified depending on the stream properties. Two kinds of deadlines are considered for a specific stream: *application deadline*, d, which is the one set by the application through the driver interface or other operations, such as QoS negotiation; *scheduling deadline*, l, which is a virtual deadline internally used by the disk scheduler and computed by the server's scheduler, so that $l \leq d$. The computation of the virtual deadline l is different for each kind of stream. For a best-effort request, l is originally set to a very large value that can be dynamically modified. For a statistic stream, l is the same as its actual deadline d. A dynamic priority is then computed, based on the former parameters, and assigned to the request.

The intuition after our policy is that if the density of real-time streams is high, more disk serving time should go toward real-time streams. Otherwise, best-effort requests could be served, because the deadlines of the currently most urgent real-time streams are not likely to be missed even if the disk server turns to serve some best-effort requests firstly. As the results shown in section 4 corroborate, our scheduling architecture has two major features related to other disk scheduling schemes. First, the *deterministic streams deadlines can be always met* for streams admitted. Secondly, *the average waiting time for best-effort requests is small.*

Obviously, scheduling deterministic streams with high priority means again that some statistic streams would miss their deadlines. An alternate approach can be used to reduce the number of missed deadlines: temporal degradation of the QoS of some streams. Usually some statistic applications, such as video-conferencing, can degrade temporally its QoS to benefit other deterministic applications, such as telesurgery. To use this scheme, each stream should specify the average QoS required and the percentage of temporal degradation during a maximum time. Then, the priority of its requests could be reduced to satisfy the new requirements.

3.2 Metascheduler Architecture

The meta-scheduler (Figure 1) is a coordinator, whose function relies in three aspects: decomposing the incoming streams and dispatching them to the corresponding disk servers; gathering portions of data from different disk servers and sending them back to the applications; and coordinating the service to a specific stream among different disk servers. The first two functions are included into the parallel disk system interface, while the third one is internal and acts as an intelligent inspector among different disk servers. When a new stream arrives, it is decomposed into substreams by the meta-scheduler and each substream is sent to the appropriated disk server. The meta-scheduler gathers the information from the servers and returns to the application the status of the admission tests: successful if all disk servers admitted the new stream, failed in other cases. When successful, the meta-scheduler asks the disk servers to commit the resources. A problem incurred with stream distribution is that some substreams could be successful while other could fail. The meta-scheduler gathers the status of the substreams and, if there are some deadline misses and the QoS is below the required, notifies a failure to the application. Moreover, it notifies to the remaining involved servers to abort the stream and to release the budget allocated to this stream. To accomplish it, each stream is assigned a unique id number, which is shared by all of its sub-streams , and inserted in a stream dispatching table. Whenever a disk server fails to serve a sub-stream, the meta-scheduler is informed. According to the unique id number, the meta-scheduler changes the status of all the sub-stream corresponding to this stream to *failed*, informing other disk servers of this situation. As a result, all the pending sub-streams of this stream are deleted from the queue in each disk server, and the resources are freed. This policy avoids wasting resources on failed streams, transferring those resources to other *successful* streams.

4 Performance Evaluation

The performance results presented in this section were collected on a Silicon Graphics Origin (SGO) machine located at the CPDC of Northwestern University, and on a Pentium biprocessor, with four disks, located at the DATSI of Universidad Politénica de Madrid. The SGO was used to test the scalability of our solution on a parallel system including several servers. To test the performance and behavior of our solution, a video server and several remote clients were used. The video server transmitted several movies attending to client requests. The duration of the movies were 30 minutes approximately.

To test the features of the scheduler, with different scheduling policies, four parameters were studied and evaluated:

1. *Disk bandwidth.* Several processes executing simultaneous read/write operations.
2. *Disk answer time.* Several video streams executing read operations simultaneously to best-effort requests.

3. *Single server performance.* Several video clients accessing MPEG videos through a 100 Mb/s Ethernet.
4. *Parallel server performance* Several video clients accessing MPEG videos sorted on several servers through a high bandwidth bus.

The former experiments were executed using several popular disk scheduling policies (FIFO, SCAN, CSCAN, EDF, and SCAN-EDF) to compare them with our *2-Q* algorithm.

Figure 2 shows the aggregated bandwidth for a single disk using several scheduling policies. It shows that the bandwidth is higher for *2-Q* than for the other algorithms. Specifically, the results of *2-Q* are better than those of *CSCAN*, which is typically used in disk drivers. These results can be explained because the deterministic requests are prioritized over the best-effort ones when using our scheduler. By doing that, more contiguous I/O requests are sent to the disk, and the seek time is reduced.

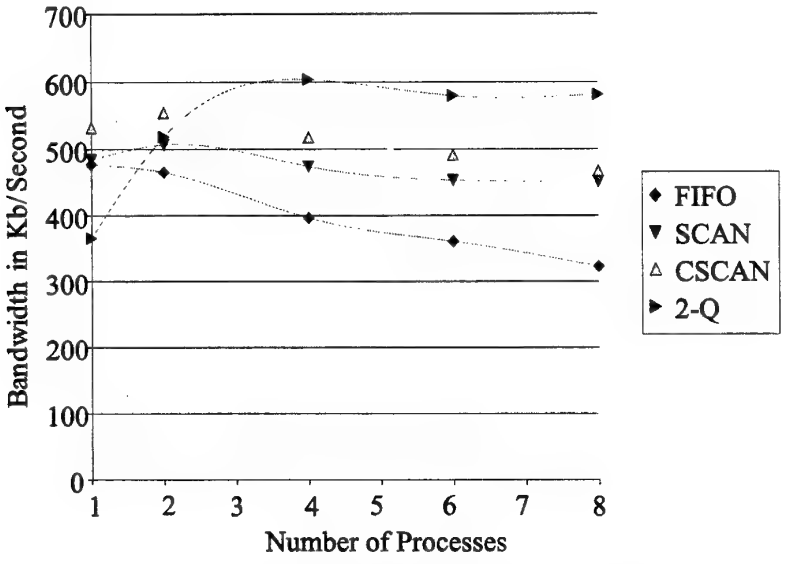


Fig. 2. Disk bandwidth with different scheduling policies

Moreover, as shown in figure 3, our scheduler has a better response time for best-effort requests than those of *EDF* and *SCAN-EDF*, typically used in continuous media systems. These results can be explained because of the *opportunistic* behavior of our disk driver scheduler, that serves best-effort requests in CSCAN order whenever it has some free time in a round. It not only reduces the answer time, but also minimizes the average seek time for best-effort requests. Those features are not present in EDF or SCAN-EDF.

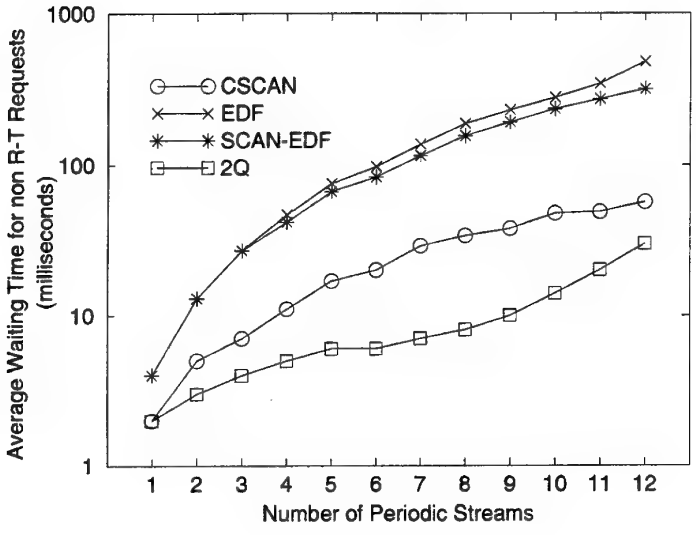


Fig. 3. Response Time for Best Effort Requests.

To test the performance of a single server, we used several video clients accessing MPEG videos through a 100 Mb/s Ethernet. All the videos were stored on a single server (the Pentium machine). Several priority levels were used to accomplish this test. Figure 4 shows that dynamic priorities provide better performance and QoS than policies with fixed priority.

To evaluate the behavior of our parallel disk server scheduler, an increasing workload was applied to a parallel disk server whose number of disk servers was varied from 1 to 16. This experiment was executed on the SGO machine. We wanted to measure the maximum number of statistic streams (periodic) served before having a deadline miss. Figure 5 shows the results of test 4. The workload was composed of deterministic sporadic streams, statistic periodic streams, and best-effort requests. As can be seen, the *2-Q* algorithm always provides the same or better results than the others. That means that *2-Q* can serve more statistic clients before having a deterministic deadline missed.

5 Summary and Concluding Remarks

In this paper, we presented a solution for the scheduling problem in a parallel integrated storage platform which can satisfy the requirements of real-time applications, multimedia systems, and traditional best-effort applications. First, we motivated the need of such a solution, then presented the architecture used in our *2-Q* scheduling architecture. *2-Q* has a hierarchical two-level architecture

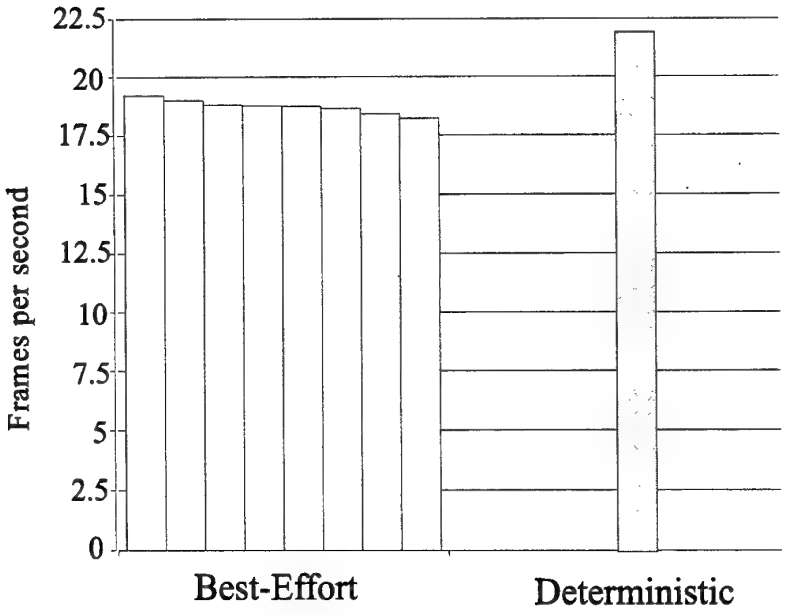


Fig. 4. Influence of priority on the number of frames per second

where the upper level, or server level, is divided in three queues for deterministic, statistic and best-effort requests, each one using a scheduling algorithm specific for that server. The solution proposed for one disk served was generalized for a parallel disk server by using a meta-scheduler to control the achievement of the deadlines of a parallel stream.

Performance evaluations, made on a Pentium biprocessor and a SGO machine, demonstrate that our scheduling architecture is adequated for handling stream sets with different deterministic, statistic, or best-effort requirements. Moreover, it maximizes the bandwidth of the disk, while minimizing the average answer time for best-effort requests. The results of the evaluation of the parallel disk scheduling architecture demonstrates that the fact of satisfying the deterministic requested does not diminished the scalability of the solution when several disks are used.

References

1. Beck M., Bhme H., Dziadzaka M., Kunitz U., Magnus R., Verworner D. Linux Kernel Internals. Second Edition. Ed: Addison-Wesley, 1998
2. Carretero,J., Zhu,W., Shen, X. and Choudhary, A. MiPFS: A Multimedia Integrated Parallel File System International Joint Conference on Information Systems, October 23-28, 1998. Research Triangle, Raleigh, North Carolina, USA.

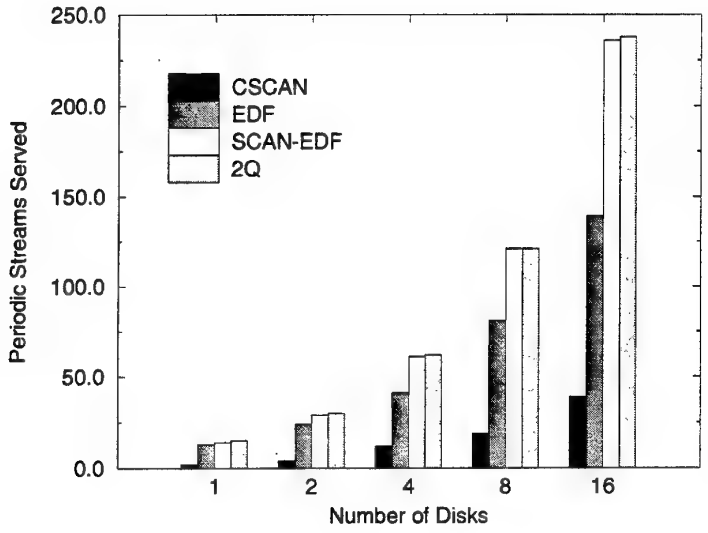


Fig. 5. Number of Periodical Clients Served.

3. Carretero J., Zhu W., and A. Choudhary A. Design and Evaluation of a Multimedia Integrated Parallel File System IEEE International Conference on Multimedia Computing and Systems ICMCS'99, Florence, Italy, June 7-11, 1999.
4. Carretero J., Zhu W., and Choudhary A. Hierarchical Scheduling for Disk I/O in an Integrated Environment ISCA 14th International Conference on Computers and Their Applications, Cancn, Mexico, April 7-9 1999.
5. S. Chaudhry and A. Choudhary. Scheduling algorithms for guaranteed service. Technical report, CASE Research Center, Syracuse University, August 1996.
6. Z. Deng and J. W.-S. Liu. Scheduling real-time applications in an open environment. In *Proc. IEEE Real-Time Systems Symposium*, pages 308–319, San Francisco, California, December 1997.
7. D. Makaroff G. Neufeld and N. Hutchinson. Design of a variable bit-rate continuous media server for an atm network. In *Proc. of the IST/SPIE Multimedia Computing and Networking*, San Jose, CA, USA, Jan 1996.
8. J. Gemmel. Multimedia network file servers: Multi-channel delay sensitive data retrieval. In ACM, editor, *Proceedings of the ACM Multimedia'93*, pages 243–250, 1993.
9. S. Ghandeharizadeh, S.H. Kim, and C. Shahabi. On configuring a single disk continuous media server. In *ACM SIGMETRICS '95*, pages 37–46, 1995.
10. J.L. Hennesy and D.A. Patterson. *Computer Architecture: A Quantitative Approach.* Morgan-Kaufmann, 2nd edition, 1995.
11. I. Kamel and Y. Ito. Disk bandwidth study for video servers. Technical Report 148-95, Matsusita Information Technology Laboratory, April 1996.

12. H. Kaneko, J.A. Stankovic, S. Sen, and K. Ramamritham. Integrated scheduling of multimedia and hard real-time tasks. Technical Report UM-CS-1996-045.ps, Computer Science Department. University of Massachusetts, August 1996.
13. J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm - exact characterization and average case behavior. In *Proc. of the IEEE Real-Time System Symposium*, pages 166–171, 1989.
14. C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real time environment. *Journal of the ACM*, 20(1):46–61, January 1973.
15. Sriram Rao Prashant Shenoy, Pawan Goyal and Harrick Vin. Symphony: An integrated multimedia file system. In *Proc. of the SPIE/ACM Conference on Multimedia Computing and Networking (MMCN'98)*, San Jose, CA, USA, 1998. Also available as Technical Report TR-97-09, Department of Computer Sciences, Univ. of Texas at Austin.
16. A.L.N. Reddy and J. Wyllie. I/O issues in a multimedia system. *IEEE Computer*, pages 69–74, March 1994.
17. C. Ruemmler and J. Wilkes. Multimedia storage servers: A tutorial. *IEEE Computer*, 27(3):17–28, March 1994.
18. Rusling D. A. The Linux Kernel. Linux Documentation Project, 1998
19. M. Sohn and G.Y. Kim. Earliest-deadline-first scheduling on nonpreemptive real-time threads for continuous media server. In *Proc. of the High-Performance Computing and Networking'97*, 1997.
20. H.M. Vin, P. Goyal, and A. Goyal. A statistical admission control algorithm for multimedia servers. In ACM, editor, *In Proceedings of the ACM Multimedia'94*, pages 33–40, San Francisco, October 1994.

A Parallel VRML97 Server Based on Active Objects

Thomas Rischbeck and Paul Watson
{Thomas.Rischbeck* |Paul.Watson}@ncl.ac.uk
Department of Computing Science, University of Newcastle
Newcastle Upon Tyne, NE1 7RU United Kingdom

April 1, 2000

Abstract

The Virtual Reality language, VRML97, allows the creation of dynamic worlds that respond to user interaction. However, the serial nature of current VRML browsers prevents the full potential of the language from being realised: they do not have the power to support huge, complex worlds with large numbers of interacting users. This paper presents the design of a scalable, parallel VRML server that has been built to overcome this barrier. The server distributes the task of storing and computing the changing state of the world across a set of nodes. Clients connect to the server and receive information on their current view of the world, which they can then render. The parallel server is implemented in Java, utilising a new, active object model called SODA (System Of Dynamic Active Objects) that is also described in the paper.

Topics covered: Virtual Reality, Parallelism, Active Objects.

1 Introduction

The Virtual Reality Modeling Language (VRML) [CB97] allows three-dimensional worlds to be described in a platform-independent notation. A world description can be downloaded over the Internet into a VRML browser that allows the user to explore the world by navigating around inside it. The first version of VRML supported only static, unchanging worlds. However, the later VRML97 [CBM97] standard supports "moving" worlds that can be both dynamic, and responsive to interaction between a user and the world. It is therefore possible to envisage the creation of huge, complex worlds

*candidate for best student paper award

with thousands of interacting users. For example, models of cities could be built to include moving vehicles as well as the buildings. In the future, with advances in traffic sensing technology, it may even be possible to build models of real cities that show accurate traffic flows in real-time.

Despite the obvious potential, VRML worlds available on the Internet have so far been relatively small, with little movement or interaction. This is due to the nature of the VRML usage model, in which the description of the VRML world is downloaded into the users browser and run locally. This causes several problems:

- downloading the complete world description to the VRML browser takes a long time for large, complex worlds.
- large worlds can have huge memory demands that a user's desktop machine may not be able to satisfy.
- the user's desktop machine must both continually update the state of the world (as objects move and the user interacts with it), and also render a view of the world in the browser window. The processing power required to do this may, for large complex worlds, be greater than the user's machine can provide, and so the user may see slow, jerky movement.
- because the world runs locally, in the user's browser, there is no possibility of interaction between different users in the same world. This precludes both direct interaction between users who meet each other in the world, but also indirect interaction, for example one user building a structure that can be seen by others.
- it is much more difficult to arrange for the world to change in response to external events. For example, if a virtual world models the current state of part of the real world (e.g. traffic flow in a city, footballers playing on a pitch) then we would wish to move the virtual world's objects to reflect the real-time changes to the objects in the real world.

This paper presents the design of a system that has been built with the aim of directly addressing these problems, and so supporting huge, complex worlds filled with large numbers of interacting users.

Our design provides a client-server implementation of VRML, in which a server holds the state of the virtual world. The state changes over time as objects move, and users interact with the world. Many clients, each running a VRML browser, can connect to the one server and so share a single world, interacting with each other as required. When a client first connects to the server, it receives only the set of geometric objects that are visible from its initial starting position. This minimises download time. As the viewer moves and interacts with the world, it receives from the server updates

to the position of any geometric objects in the field of vision that have changed, and also information on any geometric objects that have become visible. Therefore, the client has little work to do other than render its view of the world. It makes sense to leave rendering to the client as PCs and workstations have powerful graphics cards dedicated to this task.

A consequence of this architecture is that when supporting complex systems with many users, the server will have much work to do. We do not want to just move the system bottleneck from the client to the server, and so we have designed a scalable, parallel VRML server that allows the work of computing the state of the world, and supporting clients, to be spread over a set of nodes. The parallel server is implemented in Java, utilising a new, active object model called SODA (System Of Dynamic Active Objects).

2 VRML Execution Model

In this section we give a basic overview of the VRML97 execution model, focusing on those aspects that are important for the design of a parallel implementation.

2.1 Basic Terminology

A VRML world is described in terms of an acyclic and directed *scene graph* populated with *nodes* of various types and defined in one or more textual files. The scene graph is hierarchically structured through *grouping nodes*, which may contain other nodes as descendants; a *Transform* node, for example, describes geometrical transformations that influence all descendant geometric nodes.

VRML97 has 54 pre-defined node types, abstracting from various real-world objects and concepts. They reach from basic shapes and geometry, over grouping nodes and light sources to audio effects. Every node type stores its state in one or more typed fields. Examples are a Transformation node's translation, orientation and scaling fields, a *Material*'s colour and a *SpotLight*'s intensity.

Other nodes are responsible for driving and controling the dynamic behaviour of a scene, namely *Sensor* nodes, various *Interpolator* nodes and *Script* nodes.

Sensor nodes are distinguished in that they are reactive to the passage of time or to user interaction (e.g., "touching" of objects, user proximity, etc.). If stimulated, a sensor node dispatches an *event* on one or more of its *eventOut* fields (e.g., a *TimeSensor* can send an event at regular time intervals on its *cycleTime* eventOut field). All events comprise a typed value and a *timestamp*, which is determined by the sensor's trigger time. Events can be propagated from the producing eventOut field along *routes* to the

eventIn fields of other nodes. Upon receiving an event, nodes may change their state, perform event processing or generate additional events. Routes are determined by the edges of a directed *routing graph* that mediates one-way event notification between nodes. The structure of this routing graph is completely orthogonal to the scene graph hierarchy.

Event processing at a node can take the form of simple key-framed interpolation as this is done by *interpolator* nodes. *Script* nodes are more powerful in that they allow arbitrary, author-defined event processing and generation. A world author can associate a Java or JavaScript function with each eventIn field.

2.2 Event Cascades

When processing an event that it has received, a node may not only change its state, but also generate additional events. In this manner, a single sensor event can trigger an *event cascade* involving a subset of the routing graph's edges. All events in an event cascade are considered to occur simultaneously and therefore carry the same timestamp as the initial sensor event. To prevent infinite loops in a cyclic routing graph, every eventOut is limited to at most one event per timestamp[1].

Ideally, all events would be processed instantaneously in the order that they are generated. However, in a real implementation, there will always be processing delays. Furthermore, sensor nodes may generate events more frequently than the resulting event cascades can be evaluated. The VRML97 specification addresses this issue by requiring implementations to evaluate events in increasing order of timestamps. This ensures that implementations produce deterministic results.

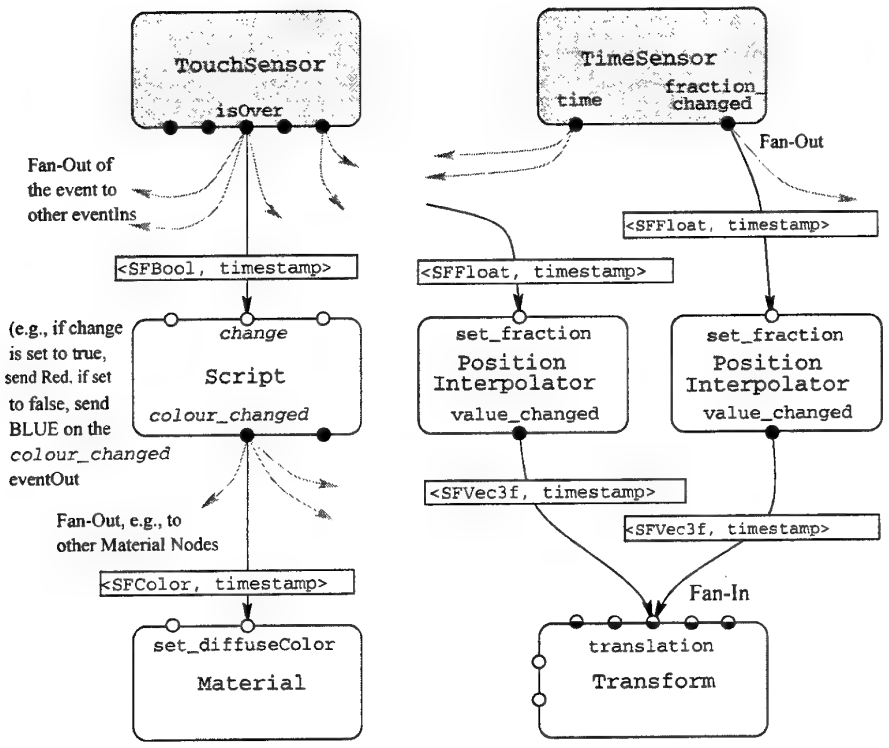
Multiple eventOuts may route to the same eventIn, in what is called a *fan-in* configuration. If events with the same timestamps arrive, they "shall be processed, but the order of evaluation is implementation dependent." ([CBM97], paragraph 4.10.5)

2.3 Discrete and Continuous Events

Most events produced during world execution are *discrete*: they happen at well-defined world times, e.g. as determined by the time of user interaction. However, TimeSensor nodes also have the capability to model continuous changes over time: A browser generates sampling events on the *fraction_changed* and *time* eventOut fields [2] of TimeSensors. The sampling

[1] Called *loop breaking rule* in VRML ([CBM97], paragraph 4.10.4.

[2] *fraction_changed* describes the completed fraction of the current cycle as a float value in the interval $[0, 1]$; *time* sends the absolute time in seconds since Jan 1, 1970, 00:00:00 GMT as a floating-point value.



(a) Discrete Initial Event

(b) Continuous Initial Events driving an Animation

Figure 1: Simple Event Cascades for Different Sensor Events (circles depict different field types: filled↔eventOut, empty↔eventIn, semi-filled↔exposedField)

frequency is implementation dependent, but typically, samples would be produced once per *frame*—e.g., once for every rendering of the user's view on the world.

Additionally, VRML requires continuous changes to be up-to-date during the processing of discrete events. i.e., "continuous changes that are occurring at the discrete event's timestamp shall behave as if they generate events at that same timestamp" ([CBM97], paragraph 4.11.3.).

Example 1 Figure 1(a) depicts a simple event cascade. The TimeSensor's isOver eventOut sends **<true, touchTime>** whenever the user moves the pointing device over its geometry and **<false, retractTime>** upon retraction.

These events are routed to a Script node—amongst other destinations

in a fan-out configuration—which performs author-defined event processing. In this example resulting in colour value being sent to a Material node. A world author might employ such a scenario to provide user feedback for the touch of a button.

Example 2 The TimeSensor in figure 1(b) produces continuous events containing a number in the range $[0, 1]$ on its *fraction_changed* field with the passage of time. These continuous events are passed to a PositionInterpolator that animates the *translation* vector of a Transform node. In this way, VRML provides support for linear key-framed animation. A fan-in situation can arise for the Transform node, if both PositionInterpolators send events with identical timestamp.

2.4 Sequential Implementation

Algorithm 1 shows the pseudo-code algorithm of a typical VRML97 browser. If no discrete events are scheduled, continuous events are sampled as quickly as possible, adapting the sampling frequency to hardware capabilities. This event evaluation is alternated with frame rendering of the new geometric layout.

Scheduled discrete events force the evaluation of all continuous events at that same time (see up-to-date requirement above). If any discrete events have not yet been evaluated, no rendering takes place.

Algorithm 2 shows the evaluation of the event cascade for each initial (sensor) event C_i or D_i (mapped to E). The loop breaking rule prohibits cyclic loops by limiting each eventOut to only one event per timestamp. Otherwise, R' contains all edges of the routing graph pointing out of E. R's fan-out destinations In_i are evaluated in turn. Possibly, event processing at the destination In_i may result in the creation of further events E'_{ij} and therefore recursive invocations of algorithm 2 until the complete event cascade is evaluated.

Algorithm 2 represents only *one* possible way of ordering event processing of conceptually simultaneously occuring events for sequential execution. Beyond the requirement that events be evaluated in timestamp order, VRML does not specify any ordering of event processing. I.e., the evaluation order of branches in a fan-out configuration as well as for eventIn processing at fan-in nodes is implementation dependent.

3 Opportunities for Parallelism

As worlds become more complex, the main loop of algorithm 1 takes more time, which can result in a reduced sampling frequency for continuous events, and therefore jerky scene updates. Further, the system may become oversaturated with discrete events if they are generated more frequently than

Algorithm 1 Sequential VRML97 Pseudocode

lasttime $\leftarrow 0$;
loop
  now $\leftarrow Browser.getWorldTime()$;
  if any discrete sensor eventOuts S_i scheduled with $lasttime < t_{E_i} \leq now$, e.g., asynchronous user input, or finished `TimeSensor` cycle **then**
    $t_D \leftarrow$ time of most imminent S_i;
    $D \leftarrow \{D_j | t_{D_j} = t_D\}$;
    $C \leftarrow$ sample of all continuous eventOuts at time t_D;
    evaluate event cascade for each $C_i \in C$; /*algorithm 2*/
    evaluate event cascades for each $D_j \in D$; /*algorithm 2*/
    $lasttime = t_D$;
  else
    $C \leftarrow$ continuous events sampled from all active and enabled `TimeSensors` at time *now*;
    evaluate event cascades for each $C_i inC$; /*algorithm 2*/
    $lasttime = now$;
    rendering of the new geometric world layout;
  end if
end loop

Algorithm 2 Event Cascade Evaluation for a sensor Event E

if eventOut E has already 'fired' for time t_E **then**
  stop; loop breaking rule
else
  $R' \leftarrow \{(Out, In_i) \subset R | Out = E\}$
  process all In_i, potentially generating a set of new events E'_{ij} for each In_i;
  evaluate event cascades for all E'_{ij} produced by using this algorithm recursively;
end if

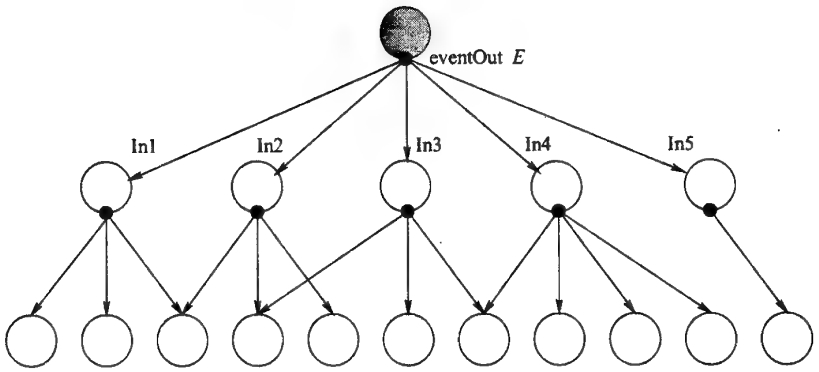


Figure 2: Parallel Evaluation of Single Event E. All events have the same timestamp t_E.

the system is able to evaluate their event cascades. In this section we examine opportunities for tackling these problems by parallelising the VRML execution model.

3.1 Parallelism Within a Single Event Cascade

In algorithm 2, if a single initial sensor event E has a fan-out configuration, all eventIn fields In_i linked to it can be processed in parallel (see figure 2). Recursion may lead to an even higher degree of parallelism. This is possible without affecting VRML97 semantics, as no evaluation order for fan-out events In_i is defined. As event notification is the sole communication mechanism between nodes, there can be no undesirable interference between two execution paths.

Due to fan-in configurations, two execution paths might reunite at one, common node. To avoid unwanted side effects in updating the node's private fields, it is paramount that event processing is performed sequentially at the node. I.e, some form of synchronisation is necessary for incoming events—for example a queue which buffers pending requests for processing.

Widely branching event cascades produced by single sensor events may exhibit high degrees of parallelism. The grain size is only determined by the complexity of event processing in the participating nodes.

3.2 Parallelism between Event Cascades

If several initial sensor events are scheduled with the same timestamp, VRML treats them as if they are members of the same event cascade. Fan-ins of events with the same timestamp are allowed and ordering is in the implementation's responsibility. Multiple writes to a single eventOut field

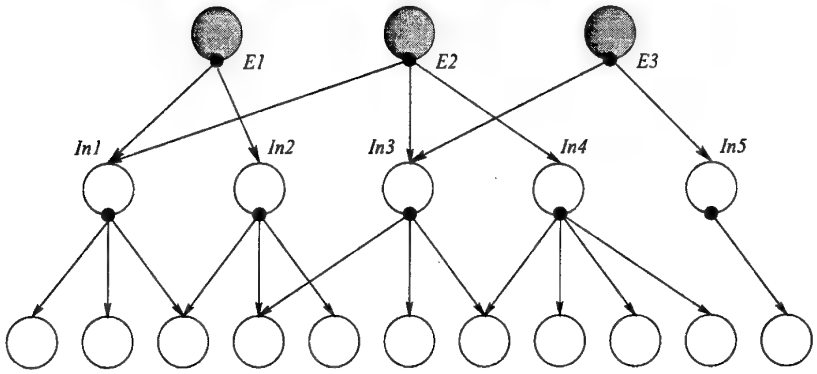


Figure 3: Event Cascade with several Initial Events E_i all of which have the same timestamp t_E.

are inhibited to satisfy VRML's loop breaking rule.

All events D_j and C_i scheduled in the main loop of algorithm 1, can therefore be evaluated in parallel[3] (see figure 3), with the same restrictions for fan-in as discussed in 3.1.

3.3 Routing Graph Partition

Parallelizing event cascades with different times is more intricate. The VRML specification requires that events be evaluated in timestamp order. Parallel processing of event cascades for different timestamps could result in a node processing events out-of-order and thus violating the VRML specification.

However, if we can identify disjoint partitions of the routing graph, then parallelism can be exploited. The routing graph is defined as a structure connecting eventIn fields to eventOuts. We define a partition as all routes that are reachable from any node, following all eventOuts at the destination node.

For disjoint partitions, event cascades with different times can run in parallel, as no interference can take place. Within a partition, such cascades have to be serialised in order of timestamps. This ignores the issue of a dynamically changing routing graph[4], which would require the dynamic examination of the routing graph.

This approach might minimally influence the perception of the world: users may notice the effects of out-of-order changes to visible nodes. However, we can assume that such differences in timestamps would only be in the

[3] i.e., by kicking off several instances of algorithm 2 for each event

[4] Script nodes in VRML might be programmed to change the topology of the routing graph dynamically

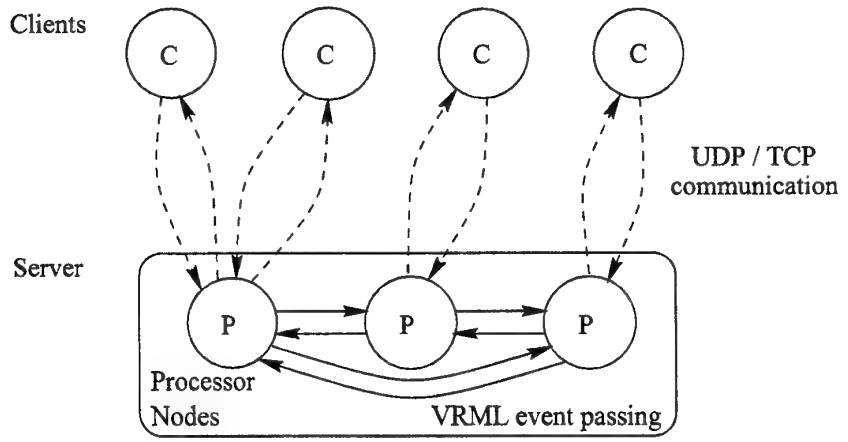


Figure 4: Parallel VRML Client-Server Model

range of a few milliseconds, and this is therefore unimportant for almost all worlds. Causally related behaviour will always be presented in the correct order as this is sequentialised through dependencies in the routing graph.

3.4 Further Parallelism

Beyond the above we identified further opportunities for parallelism are as below:

Evaluation of Sensor Nodes can be done in parallel if their required sensor information is available (e.g. current time, user location, etc.). Sensor nodes may then register discrete events with a Scheduler.

Scheduler The whole of algorithm 1 may be replicated for each partition of the routing graph. Again, synchronised time must be available at each location.

4 Implementation

The following gives a quick overview of the System Of Dynamic Active Objects (SODA), which is used as programming model and runtime system for implementaiton of the VRML server. A more in-detail description of SODA will soon be available as a technical report.

4.1 Active Objects Programming Model

SODA adopts a programming model of coexisting active and passive objects, similar to ProActive [CV98]. Active objects encapsulate a concurrent

activity and have a queue to buffer method invocations for serial processing by this activity. Neither explicit thread programming nor intra-object synchronisation code is required. Active objects are globally addressable and passed by reference. In contrast, passive objects can only exist privately to an active object and consequently have pass-by-value semantics.

Programs consist of a collection of active and private objects, which, if distributed over several processors, proceed in parallel. Unlike ProActive, active objects are fully location transparent and do not require explicit mapping to a parallel platform.

Method calls are by definition non-blocking. The callee can proceed without waiting for the caller to return. Upon termination of the method the callee may hand back results in a future mechanism, similar to ProActive [CV98]. In addition, SODA defines *Collectors* as an additional inter-object synchronisation mechanism. These are capable of detecting termination of a method call together with its complete cascade of subcalls and inform an arbitrary active object about this condition.

4.2 SODA Runtime

The SODA runtime system is characterised by several key features:

Dynamic Load Balancing Through Active Object Migration. Transparently to the programmer, the SODA runtime system is responsible for spreading out active objects during runtime with the aim of dynamically maximising processor utilisation for the overall system. This is important where active objects have relatively high fluctuations in their resource requirements.

Active Object Multiplexing on Threads. Active objects may be multiplexed onto threads. This prevents thread flooding and encourages programmers to use relatively many active objects without worrying about negative performance impact.

True One-Way Method Calls. Many related systems (e.g. JavaParty [PZ97] and ProActive) use RMI as transport protocol for method invocations on active objects. This has disadvantages for modelling an active objects model. This has the following disadvantages:

RMI, being designed as a client-server protocol does not provide an asynchronous communication mechanism. Related systems based on RMI overcome this by creating an additional thread at the caller to wait for method call termination, which incurs performance penalties. Additionally, RMI opens a TCP/IP port for every remote object allocated. This is not optimal for a large number of active objects.

In contrast, SODA uses a socket-layer communication protocol to implement one-way calls. At most one TCP/IP connection is established

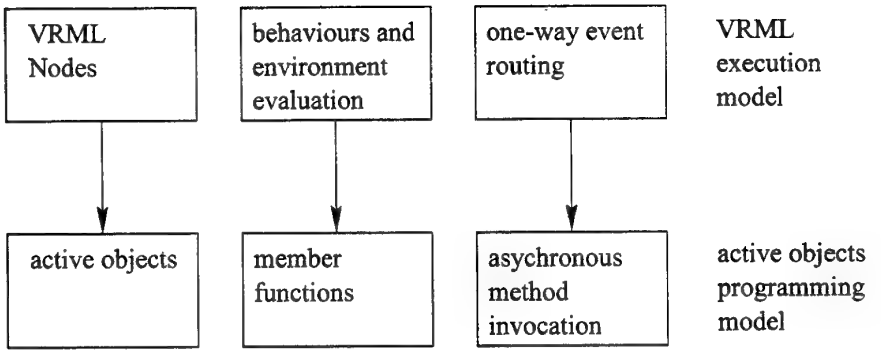


Figure 5: Mapping VRML onto Active Objects

between every pair of hosts. SODA also exploits "unexpected locality" of active objects [PZ97]. I.e., no expensive loopback communication takes place if caller and callee reside on the same JVM.

4.3 Mapping onto Active Objects

VRML nodes and SODA active objects share many commonalities. Following the object-orientation paradigm, communication among VRML nodes can only take place through a well-defined interface. In both systems, incoming messages trigger the asynchronous execution of member functions.

We applied a mapping between components of the two systems (see figure 5) as follows: VRML nodes can be directly mapped onto active objects. Those nodes may then perform parallel event generation or processing, which is the mainstay for parallel event cascade evaluation.

Asynchronous VRML event passing is mapped onto the asychronous and one-way communication semantics of SODA. Figure 5 shows how all elements of the VRML execution model find a valid equivalent in SODA.

Distributed Schedulers (see 3.4) can be implemented as active objects. SODA's collector mechanism is used to inform the scheduler object about termination of an event cascade.

Altogether, our experience from creating a parallel VRML prototype showed that SODA offers a good match for supporting VRML nodes, behaviours and the routing mechanism.

4.4 Client-Server Communication

Client-server communication is based on a light-weigt datagram protocol for performance reasons. Clients may communicate with any server node to send the user's position and to receive information about objects within

their respective view volume (see figure4). Client-server communication can be reduced by performing server-side culling or area of interest management [SZ99].

4.5 Preliminary Performance Results

For preliminary performance testing we used a network of four Pentium II 233 Mhzworkstations, running Linux 2.2.5 and Sun's JDK 1.2.2, as our server. The machines were networked via fast ethernet connections. The client was running Windows NT on a Pentium II-300 without hardware graphic acceleration, connected via a only 10Mbit Ethernet on the same LAN. We used CosmoPlayer 2.1 as VRML browser.

As a simple experiment, we created a scene graph containing a primitive geometric objects. The translation of each object was influenced through a Script node driven by a separate TimeSensor. To get results for varying complexity of event processing, the granularity of the Script node was changed over a series of tests by cycling through an empty loop. The experiment was performed for one and four TimeSensor-Script-Object triplets, once locally (in the browser's JVM), then remotely (on the server). The first remote run uses only one server node, while the second employs all four.

Script Granularity	0	10^4	10^5	10^6	10^7	10^8
local 1 TS $\rightarrow$ 1 Scipt	40.3	39.3	37.8	27.8	7.69	1.29
remote 1 TS $\rightarrow$ 1 Scipt	89.3	88.7	88.9	87.9	13.6	1.14
local 4 TS $\rightarrow$ 4 Scipt	28.8	28.1	25.4	12.7	2.18	0.2
remote 4 TS $\rightarrow$ 4 Scipt	27.6	25.7	25.7	25.8	16.6	5.5

Those values are slightly influenced by fluctuations on the shared network connecting client and server. However, it is interesting to notice that even without parallelism the client-server approach gives much higher update rates. This can be related to the cost of software rendering at the client, i.e. the two machines share world evaluation and rendering.

For four triplets, the update rate is similar when scripts have low event procesing costs. As those get higher however, the performance of remote parallel execution becomes visible.

5 Conclusion and Further Work

This paper has presented a novel, scalable, client-server based approach to implementing complex virtual worlds with many interacting users. The clients that browse the world are protected from the costs required to support a large, complex world by the server, which carries the burden of progressing the state of the world, and determining the fraction of the world that is visible to each client. The work of the client is restricted to receiving

updates to the part of the world that it can see, and rendering a view of this. The server is able to support large, complex worlds due to its scalable, parallel design. The paper has shown how a parallel implementation of VRML can be built without changing the semantics of the execution mechanism. The results from the prototype show that real performance gains can be achieved. Experience in building the prototype using SODA has shown the power of the active object model for parallel, object-based software design.

Future work will include tuning the system, and analysing its ability to scale up to huge, complex worlds, with many users, running on a larger number of parallel nodes.

References

[CB97] Rikk Carey and Gavin Bell. *The Annotated VRML 2.0 Reference Manual.* Addison-Wesley Publishers, 1997.

[CBM97] Rikk Carey, Gavin Bell, and Chris Marrin. Iso/iec 14772-1: 1997 virtual reality modelling language (vrml97), 1997.

[CV98] Denis Caromel and Julien Vayssiere. Towards seamless computing and metacomputing in java. In Geoffrey C. Fox, editor, *Concurrency: Practice and Experience*, volume 10, pages 1043–1061. Wiley & Sons, 1998.

[PZ97] Michael Philippsen and Matthias Zenger. Javaparty – transparent remote objects in java. In Geoffrey C. Fox, editor, *Concurrency: Practice and Experience*, volume 9, pages 1225–1242. Wiley & Sons, 1997.

[SZ99] Sandeep Singhal and Michael Zyda. *Networked Virtual Environments.* ACM Press, 1999.

Cellular Automata: Applications

Dietrich Stauffer

Institute for Theoretical Physics, Cologne University, D-50923 Köln, Euroland
stauffer@thp.uni-koeln.de

Abstract. A review will be given on the simulation of large simple cellular automata with up to one million times one million elements. Here each site of a large lattice carries a binary variable the orientation of which depends on its lattice neighbours orientation at the previous time step. Single-bit processing allows for high speeds (except for probabilistic rules) and saves memory. Geometric parallelization is easy since only a small amount of message passing at predictable times is required. Applications will emphasize biology: Game of life, ageing, sex.

1 Introduction

Parallel computing for cellular automata and Ising-like systems has a 30 year old history, long before real parallel machines became widespread. For each variable then can be stored in a single bit, and by logical bit-by-bit operations dealing with 32 variables simultaneously through one single 4-byte command. Many aspects of vector and parallel computing, like the division of a lattice into sublattices of checkerboard-type, were used in this way before they were used on vector computers. Physicists call this method multi-spin coding, and Prof. J.A.M.S Duarte is a Porto expert. Since my last review [1] large parallel machines became widespread, and also different applications were found.

Cellular automata are discrete in space, time, and values. For us here we assume that each site i of a large lattice carries a variable n_i which is either $+1$ or -1 (the spin language preferred by physicists), or 1 and 0 in the language of computer science which is more appropriate for multi-spin coding. The value n_i at the next time step $t+1$ is completely determined by that of its nearest lattice neighbours at time step t. These are deterministic automata; do you call a cigarette automat working if with probability $p \simeq 0.4$ it delivers a pack of cigarettes ?

The next section deals with multi-spin coding on a scalar machine, then we deal with domain decomposition of large lattices on parallel computers, and finally we summarize some applications with up to 10^{12} sites.

2 Multi-Spin Coding

Let us assume we want to study an infection process. Every site on a large lattice is either sick ($n = 1$) or healthy ($n = 0$). Sick sites infect their neigbours. Thus

$n_i(t+1)$ is sick at the next time step $t+1$, if at time t at least one of its neighbours was sick. On a one-dimensional chain these two neighbours of site i are $i-1$ and $i+1$ which means that a logical OR gives the desired result:

```
nnew(i) = n(i-1) .or. n(i+1)
```

when in Fortran n and nnew are logical arrays. Now many bits are wasted to store the one-bit variables nnew and n in one computer word. If we store 32 such variables in one computer word, we rewrite the above statement as

```
nnew(i) = ior( n(i-1) , n(i+1))
```

where ior is a bit-string command performing the logical-or operation for each pair of bits separately. Thus one command does 32 (or 64) operations in parallel on a scalar computer. In the programming language C the same bit-by-bit commands are part of the standard, using different symbols for the operations.

If of the chain we would store sites 1 to 32 in the first 4-byte integer n(1), sites 33 to 64 in n(2), etc, then the above statement would not deal with the nearest neighbours of site i. Therefore, if we use $LL = L/32$ words n(1), n(2), ... n(LL) for L sites, we store site 1 in the first word, site 2 in the second, ..., site LL in the last word, in the first bit position. Then sites $LL+1$, $LL+2$, ..., $2LL$ are stored in the second bit position of the same words n(1), n(2), ... n(LL), then $2LL+1$ to $3LL$ in the third bit position, until the last bit of the last word n(LL) is filled with site L. Then the above statement really works for $i = 2$; for the extreme words n(1) and n(LL) the left and right neighbours are n(LL) and n(1), respectively, shifted circularly to the left or right by one bit. In d dimensions, the integer array n needs a second index going from 1 to L^{d-1} as usual. Complete Fortran programs are given in [1].

In the analysis of simulated configurations it is very helpful to have a function computing the number of bits which are set. Fortunately, the late Seymour Cray was aware of that problem and gave us this function under the name `popcnt`.

These programs are also vectorized and speeds of the order of 10^9 sites could be reached already a decade ago on one vector processor. The next section describes the parallelization.

3 Parallel Computers

While multi-spin coding allows parallel treatment of 32 variables, we can get additional speed on parallel computers with many processors, distributed memory, and message passing, by using simultaneously all these processors on one large lattice. (How to do different lattices on different processors by replication presumably does not have to be explained to this conference.) Since I do not have access to a multitude of such parallel machines, I just learned the machine-dependent message passing routines on the machine for which I had the account, and since 1996 this is a Cray-T3E with 64 bits per word. Message passing commands start with `shmem`.

```
      do 3 itime=1,max
       info = barrier()
       if(node.gt.0) call shmem_get(n(1,0),n(1,Lstrip),LL,node-1)
       if(node.eq.0) call shmem_get(n(1,0),n(1,Lstrip),LL, np -1)
       info = barrier()
        do 6 j=1,Lstrip
c         periodic boundaries left and right via circular shift
          n(0,j)=ior(ishft(n(LL,j),-1),ishft(n(LL,j),63))
6         n(LL1,j)=ior(ishft(n(1,j),1),ishft(n(1,j),-63))
        nch=0
        do 7 j=1,Lstrip
        if(j.eq.2) then
        info = barrier()
        if(node.lt.np-1) call shmem_get(n(1,Lp),n(1,1),LL,node+1)
        if(node.eq.np-1) call shmem_get(n(1,Lp),n(1,1),LL,   0  )
        info = barrier()
        end if
        do 7 i=1,LL
          n1=n(i,j-1)
          n2=n(i,j+1)
          n3=n(i-1,j)
          n4=n(i+1,j)
          n5=n(i,j)
          n12=ior(n1,n2)
          n(i,j)=ior(ior(iand(n1,iand(n2,n3)),iand(n5,
     1    iand(n4,ior(n12,n3)))),iand(ior(n5,n4),
     2    ior(iand(n12,n3),iand(n1,n2))))
7         if(n(i,j).ne.n5) nch=nch+popcnt(ieor(n(i,j),n5))
        info = barrier()
        if(node.eq.0) then
        do 8 iadd=1,np-1
           call shmem_get (idummy,nch, 1, iadd)
8          nch = nch + idummy
        endif
        info = barrier()
        call shmem_get (nch,nch,1,0)
        info = barrier()
        if(node.eq.0) print *, itime,nch
        if(nch.eq.0) goto 9
3     continue
9     continue
```

Here `shmem get(target, source, length, node)` gets from the processor with number `node` the information starting there with the word `source` and

extending over `length` words in total. It stored it in the memory of the current processor, with the first memory location called target. For example, `call shmem get(n(1,0),n(1,Lstrip),LL,node-1)` gets from processor `node-1` to the present node the LL words n(1,Lstrip) to n(LL,Lstrip) and stored them in the words n(1,0) to n(LL), 0).

We divide a large $L \times L$ square lattices into N_p strips of length L and width $L_{strip} = L/N_p$. Each strip is stored on one processor; in addition, each processor stores the lowest lattice line of the strip on the upper processor, and the highest lattice line of the strip on the lowest buffer. These two buffers are updated after every iteration via the shmem get command. Thus message passing happens at predictable times, and the amount of transferred information is much smaller than the total amount of stored information provided $L_{strip} \gg 1$.

The sample program core simulated the Griffeath majority rule on the square lattice: A spin is flipped if and only if more than half of its four neighbours point into the opposite direction. Loop 7 mostly translates these words into logical statements for the bit-by-bit operations. To see if a stable configuration is reached which will remain unchanged forever we calculate the number `nch` of sites which have flipped. If this number, summed over all processors, is zero, then we can stop the iteration. This particular cellular automata rule [2] was selected since the simulation indeed comes to a stop after a moderate number of iterations, thus allowing the simulation of $L = 10^6$ with moderate computing time. (For simplicity, the program uses sequential instead of simultaneous updating. `info = barrier()` forces synchronization of all processors. There are more elegant ways than loop 8 to sum over all processors.)

4 Applications

4.1 With Multi-Spin Coding

One of the most famous applications of cellular automata are Frisch-Hasslacher-Pomeau lattice gases for hydrodynamics on a triangular lattice. However, in recent years the emphasis in this area seems to have shifted to the lattice Boltzmann equation which goes beyond cellular automata and is thus not reviewed here [3].

Immunology was simulated with cellular automata, sometimes using the above methods for huge lattices; but no consensus is evident from the literature which automata rule is best; thus we refer to [4].

(In this immunological context, a vectorization technique was developped which allows to write one Fortran program for general dimension, working e.g. on the square lattice, the simple cubic lattice, and the four- or five-dimensional hypercubic lattice, though not with one bit per spin. The number of neighbours in d dimensions is $2d$, and an inner loop over such a small number of neighbours would be very inefficient. Thus the inner loop went through all lattice sites. In the loop body, for every direction one line was added which was executed if the dimensionality d was large enough. Such if-statements again normally are

deadly for efficient vectorization. However, by denoting d as a fixed constant `idim` through `parameter(idim=3)`, the if-conditions were evaluated at compile time and the loop was efficiently vectorized.)

The Game of Life has fascinated many through its variety of configurations. It uses the 8 nearest and next nearest neighbours of the center site. If the center site is empty it becomes occupied at the next time step if and only if three neighbours are occupied; if the center is occupied it remains so for the next time step if and only if two or three of its eight neighbours are occupied. A multi-spin coding program was published by Gibbs [5] though Franco Bagnoli (priv.comm.) has a faster one. Fig.1 shows how the final density of occupied sites depends on the initial density [5]. Large lattices confirmed the theoretically expected power laws here for both low and high densities.

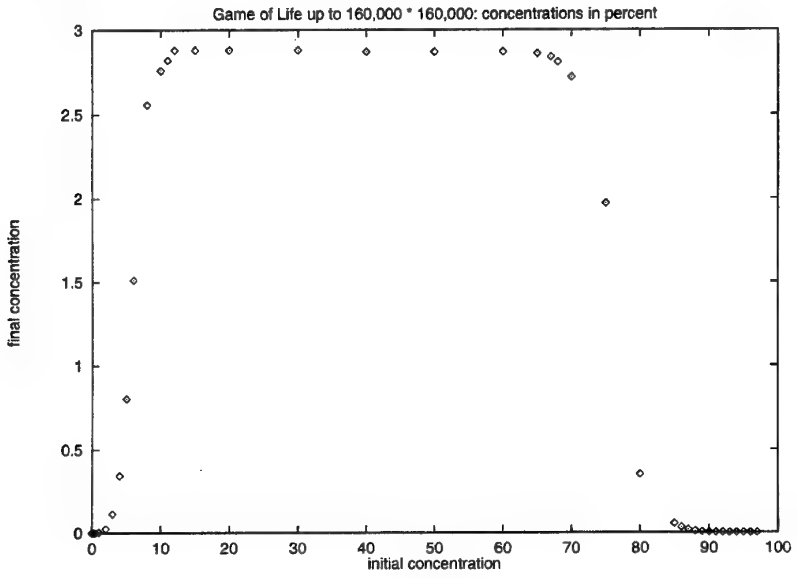


Fig. 1. Variation of equilibrium density in Game of Life with initial density if the sites initially are occupied randomly. [5]

Much simpler are the Q2R cellular automata approximating the Ising model. Each spin flips if and only if it has as many up as down neighbours. Thus, if interpreted as an Ising magnet, the spin flips if and only if such a flip does not change the energy. We have here a reversible microcanonical and non ergodic algorithm which neverless numerically gives the correct spontaneous magnetizations in two and three dimensions. The dynamical behavior, however, is not understood [6]. This algorithm is a special case of the Creutz demon method recently reviewed by Aktekin [7]; we merely let the size of the energy reservoir

of the demons go to zero. Careri [8] pointed out a possible biological application of Q2R.

4.2 Other Bit-Strings

In the above applications, all bits within one word could be treated efficiently by multi-spin coding since they all played the same role and thus were treated in parallel. This is no longer the case when the position of a bit has a special meaning. In the Penna model of biological ageing [9], the bit position corresponds to a "year" or other time unit in the life of the individual: A bit set at one year means that from this year on until the death of the individual a dangerous inherited disease affects the health; three or more such active diseases kill. Thus the bit-string in this Penna model symbolizes the survival aspects of the genome; the age of genetic death is stored in it from birth, but at present we know only those inherited diseases which have become active. (See the movie Gattaca on the question whether a future is desirable when we can interpret the human genome such that we know all these genetic defects long before they become active.)

Thus each individual, characterized by a string of 32 bits, lives until three set bits kill it. Before, if gives birth provided it has reached the minimum reproduction age; for each child a random mutation sets one of the bits to one compared with the bit-string of the parent. To avoid an exponential growth of the population, a Verhulst factor like in the logistic equation limits the population from above. Now we no longer can deal with the bit-string through multi-spin coding in the above parallel sense, since a bit for year 2 plays a different role than a bit for year 30. But the bit-handling techniques are useful for both methods.

Numerous simulations of this model, as reviewed recently [10], gave agreement with the Gompertz law of a mortality function increasing exponentially with age, or with the lifestyle of the Pacific Salmon who dies shortly after marriage. Very recently [11] it was pointed out that in experiments with flies one may have a genetically homogeneous population but still a Gompertz law whereas the Penna model would in this case predict all genetic deaths at the same age; this critique was combined with a more complicated model avoiding this disadvantage.

For parallel computing it is not only easier but also better to simulate N_p separate populations on N_p processors in parallel, than to distribute one population among the different processors. In the latter case, after some time one of the processors, which happened to have the fittest ancestors, will carry all the individuals and the others none, if no load balancing [12] is made.

The above algorithm refers to asexual cloning; sexual reproduction combines one female bit-string and one male bit-string to give the genome of the child. Compared to cloning, sexual reproduction has the immediate advantage of avoiding the dangers of a hereditary disease: If this disease is recessive, as are most mutations, and if only one of the two parents has it, then the child's health is not affected by it. In other words, sexual reproduction as opposed to cloning allows for redundant information just like back-up diskettes. If an error gets into

the hard disk (female genome), the diskette (male genome) still has the correct information. Similarly, repeated proofreading [13] avoids more error than proofreading just once. One of the main successes of the sexual Penna model was an explanation why females (as opposed to Pacific Salmon) survive menopause and why menopause exists at all for mammals [10]. It explained why men live shorter than women, opposite to the situation with birds [14]. The simulations also warn of future disappointments with medical care [15].

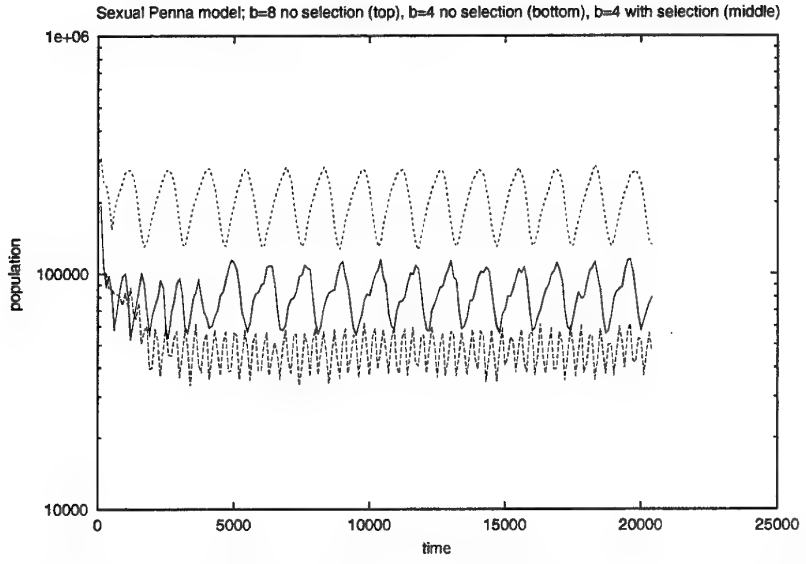


Fig. 2. Initial simulation (top), population with half the birth rate to simulate sexual reproduction (bottom), and somewhat improved results if females select only healthy partners (middle).

However, what about life-forms with two genomes but without sexual reproduction (also known as meiotic parthenogenesis). The above arguments make in this case the transmission of genetic information as reliable as in the sexual case, while the men do not get pregnant and just eat the steaks and drink the port wine away from the mothers. Why do we men exist at all ? As protection against parasites [16] ? It helps little if after thousand generations the greater genetic variety allows better adjustment to an environmental catastrophe when during the waiting time at each generation meiotic parthenogenesis wins by a factor of two compared to sexual reproduction [10, 17]. Figure 2 shows with the highest population the meiotic parthenogenesis; then for sexual reproduction the birth rate is reduced by a factor two to account for lazy men (lowest population), and finally, for the middle curve, we assume that females select only the healthier

males (less mutations) as sexual partners. We see from the figure that female selection may help, but not enough to overcome the loss of half the births.

Thus, perhaps men are an error of nature: "When God created Adam, She was only trying out."

References

1. Stauffer, D.: Computer simulations of cellular automata. J.Phys. A **24**, 909 (1991); de Oliveira, P.M.C.: *Computing Boolean Statistical Models*, World Scientific, Singapore 1991
2. Stauffer, D.: Simulation of Griffeath majority rule on large square lattice. Int. J. Mod. Phys. C **8**, 1141 (1997)
3. Boghosian, B. and Yeomans, J. : Proc. 7th Int. Conf. Discrete Simulation of Fluids, Oxford, July 1998, Int. J. Mod. Phys. C **9**, 1123-1605 (1998)
4. Lippert, K. and Behn, U.: Modelling the Immune System: Architecture and Dynamics of Idiotypic Networks, page 287 in: Annual Reviews of Computational Physics, Vol. V, World Scientific, Singapore 1997; Zorzenon dos Santos, R.M.: Immune Responses: Getting close to experimental results with cellular automata models. *ibid.*, Vol. VI, page 159 (1998)
5. Gibbs, P. and Stauffer, D.: Search for Asymptotic Death in Game of Life. Int. J. Mod. Phys. C **8**, 601 (1997); Malarz, K. et al: Some new facts of Life. Int. J. Mod. Phys. C **9**, 449 (1998)
6. Stauffer, D.: Critical 2D and 3D dynamics for q2r cellular automata. Int. J. Mod. Phys. C **8**, 1263 (1997)
7. Aktekin, N.: page 1 in: Annual Reviews of Computational Physics, Vol. VII, World Scientific, Singapore 2000.
8. Careri, G. and Stauffer, D.: Ising cellular automata for proton diffusion on protein surfaces. Int. J. Mod. Phys. C **9**, 675 (1998)
9. Penna, T.J.P.: J. Statist. Phys. **78**, 1629 (1995)
10. Moss de Oliveira, S., de Oliveira, P.M.C, and Stauffer, D.: *Evolution, Money, War and Computers*, Teubner, Stuttgart-Leipzig, 1999
11. Pletcher, S. and Neuhauser, C.: Biological aging - Criteria for modelling and a new mechanistic model. Int. J. Mod. Phys. C **11**,No.3 (2000)
12. Meisgen, F.: Dynamic load balancing for simulations of biological aging. Int. J. Mod. Phys. C **8**, 575 (1997)
13. Morris, J.A.: Medical Hypotheses 49, 159 (1997)
14. Paevskii, V.A.: *Demography of Birds* (in Russian), Nauka, Moscow 1985
15. Niewczas, E., Cebrat S., and Stauffer, D.: The influence of the medical care on the human life expectancy in 20 th century and the Penna ageing model. Theory in Biosciences, in press (2000).
16. Hamilton, W.D., Axelrod, R., and Tanese, R.: Sexual reproduction as an adaption to resist parasites. Proc. Natl. Acad. Sci. USA **87**, 3566 (1990); Howard, R.S. and Lively C.M.: Parasitism, mutation accumulation and the maintenance of sex. Nature **367**, 554 and **368**, 358 (E) (1994); Sá Martins, J. S.: Simulated co-evolution in a mutating ecology. Phys. Rev. E **61**, R 2212 (2000)
17. Stauffer, D.: Why care about sex ? Some Monte Carlo justification. Physica A **273**, 132 (1999)

The Role of Parallel Cellular Programming in Computational Science

Domenico Talia

ISI-CNR
c/o Deis, UNICAL
87036 Rende, CS,Italy
e-mail: talia@si.deis.unical.it

Abstract. Cellular automata provide an abstract model of parallel computation that can be effectively used for modeling and simulation of complex phenomena and systems. The design and implementation of parallel languages based on cellular automata provide useful tools for the development of scalable algorithms and applications in computational science. We discuss here the use of cellular automata programming models and tools for parallel implementation of real-life problems in computational science. Cellular parallel programming tools allow for the exploitation on the inherent parallelism of cellular automata in the efficient implementation of *natural solvers* that simulate dynamical systems by a very large number of simple agents (cells) that interact locally. As a practical example, the paper shows the design of parallel cellular programs by a language called CARPET and discusses other languages for parallel cellular programming.
Keywords: cellular automata, programming languages, parallel and distributed computing.

1 Introduction

Cellular automata (CA) offer a computational model that, because its simplicity and generality, has been utilized in many and disparate scientific areas such as fluid dynamics, artificial life, image processing, parallel computing, biology, economics and data encryption. The use of cellular automata has been widened by their implementation on high-performance parallel architectures that allowed for their use on solving very complex problems. Several languages and tools have been developed for programming cellular automata on sequential and parallel machines. They can support and improve the design and implementation of complex applications and systems using the cellular automata paradigm. This paper presents and discusses cellular automata programming languages and models for parallel implementation of real-life problems in computational science.

A cellular automaton consists of a lattice of *cells*, each of which is connected to a finite neighborhood of cells that are nearby in the lattice [14]. Each cell in the regular spatial lattice can take any of a finite number of discrete state values. Time is discrete, as well, and at each time step all the cells in the lattice are

updated by means of a local rule called *transition function*, which determines the cell's next state based upon the states of its neighbors. That is, the state of a cell at a given time depends only on its own state and the states of its nearby neighbors at the previous time step. Different lattice topologies (e.g., triangular, square, and hexagonal) and neighborhoods can be defined for an automaton.

Cellular automata provide a global framework for the implementation of parallel programs that represent *natural solvers* of dynamic complex phenomena and systems based on the use of discrete time, discrete space and a discrete set of state variable values. CA are intrinsically parallel and they can be efficiently mapped onto parallel computers , because the communication flow between processors can be kept low. Inherent parallelism and restricted communication are two key points for the efficient execution of CA on parallel computers. Applications of CA are very broad, ranging from the simulation of artificial life, physical, biological and chemical phenomena to the modeling of engineering problems in many fields such as road traffic, image processing, and science of materials. In the past 20 years there has been a significant increase of research activities concerning both theoretical aspects and practical implementations and use of cellular automata as a model for complex dynamics [16] [12].

In the cellular programming approach, a cellular algorithm consists of the transition function of cells that compose the CA lattice. The transition function of each cell is executed in parallel, thus the global state of the the entire automaton is updated at each iteration. For all the cells the same local rule is generally used (*homogeneous* cellular automata), but it is also possible to define some cells with different transition functions (*inhomogeneous* cellular automata).

In general, traditional languages such as C, Pascal, C++ and Fortran are used in sequential implementations of cellular automata simulations. When a parallel implementation is provided, these languages are typically used together with parallel toolkits such as MPI and PVM. An alternative to this conservative approach is to use CA languages that can express directly in their constructs the definition of CA lattices and cellular algorithms. After the program writing, a compiler translates these CA rules into a simulation program. This approach has a programming advantage offering high-level CA operations and the same CA description could possibly also be compiled onto different computers.

Our opinion is that it is necessary and very useful to develop high-level languages and tools specifically designed to express the semantics of the cellular automata computational model. In particular, the design and implementation of parallel languages based on the cellular automata model provide high-level programming tools for the development of *natural solvers* in computational science, that is scalable algorithms and applications based on a nature-inspired model such as cellular automata. In the recent years several CA-based languages have been developed and used for designing computational science applications. This paper discusses the role these languages may play in the parallel scientific applications arena. Furthermore, we show as a case study of this approach, the design of parallel cellular programs by the CARPET language and discuss other languages for parallel cellular programming such as CDL, Parcel-1, CANL, and

Cellang. Because of space limits we cannot describe in detail all the languages, therefore we discuss their main features by discussing the paramount aspects of the parallel cellular programming languages class.

The remainder of the paper is organized as follows. Section 2 introduces and discusses the main features of parallel cellular languages. Section 3 gives a brief description of the CARPET language and section 4 shows the use of CARPET for implementing scientific applications according to the cellular programming model; some figures are presented to show performance scalability. Finally, section 5 draws some conclusions.

2 Languages for Parallel Cellular Computing

The aim of the paper is to discuss how cellular programming languages can support users in the implementation of computational science applications. This class of applications require the use of high-performance computers to get results in a reasonable amount of time. For this reason we restrict the discussion on cellular languages that are have been implemented on parallel computers.

For the implementation of CA on parallel computers two main approaches can be used. One is to write programs that encode the CA rules in a general-purpose parallel programming language such as HPF, HPC++, Linda or CILK or still using a high-level sequential language like C, Fortran or Java with one of the low-level toolkits/libraries currently used to implement parallel applications such as MPI, PVM, or OpenMP. This approach does not require a parallel programmer to learn a new language syntax and programming techniques for cellular programming. However, it is not simple to be used by programmers that are not experts in parallel programming and code consists of a large number of instructions even if simple cellular models must be implemented. The other possibility is to use a high-level language specifically designed for CA, in which it is possible to directly express the features and the rules of CA, and then use a compiler to translate the CA code into a program executable on parallel computers. This second approach has the advantage that it offers a programming paradigm that is very close to the CA abstract model and that the same CA description could possibly also be compiled into into different code for various parallel machines. Furthermore, in this approach parallelism is transparent from the user, so the programmers can concentrate on the specification of the model without worrying about architecture related issues. In summary, it leads to the writing of software that does express in a natural way the cellular paradigm, thus programs are more simple to read, change, and maintain. On the other hand, the regularity of computation and locality of communication allow CA programs to get good performance and scalabiltity on parallel architectures.

Several CA programming languages such as Cellang [3], CARPET [10], CDL [6], CANL [7], Parcel-1 [13], DEVS-C++ [18], and CEPROL [8], have been designed for parallel cellular computing in the past years. These languages support the definition of cellular algorithms and their execution on different classes of parallel computers. They have several shared features such as the common

computational paradigm and some differences such as, for example, different constructs to specify details of a cellular automaton or of mapping and output visualization [17]. Many real-world applications in science and engineering, such as lava-flow simulations, molecular gas simulation, landslide modeling, freeway traffic flow, 3-D rendering, soil bioremediation, biochemical solution modeling, and forest fire simulation, have been implemented by using these CA languages. Moreover, parallel CA languages can be used to implement a more general class of fine grained applications such as finite elements methods, partial differential equations and systolic algorithms.

Here we discuss the main features of those languages. In particular, we outline the following aspects that influence the way in which CA applications can be developed on high performance architectures:

1. programming approach,
2. cellular lattice declaration,
3. cell state definition and operations,
4. neighborhood declaration and use,
5. parallelism exploitation,
6. cellular automata mapping, and
7. output visualization,

By discussing these concepts we intend to illustrate how this class of languages can be effectively used to implement high-performance applications in science and engineering using the massively parallel cellular approach.

2.1 Programming Approach

When a programmer starts to design a parallel cellular program she/he must define the structure of the lattice that represents the abstract model of a computation in terms of cell-to-cell interaction patterns. Then it must concentrate on the unit of computation that is a single cell of the automaton. The computation that is to be performed must be specified as the evolution rule (transition function) of the cells that compose the lattice. Thus, differently form other approaches, a user do not specify a global algorithm that contains the program structure in an explicit form. The global algorithm consists of all the transition functions of all cells that are executed in parallel for a certain number of iterations (steps). It is worth to notice that in some CA languages it is possible to define transistion functions that change in time and space to implement inhomogeneous CA computations. Thus, after defining the dimension (e.g., 1-D, 2-D, 3-D) and the size of the CA lattice, she/he needs to specify, by the conventional and the CA statements, the transition function of the CA that will be executed by all the cells. Then the global execution of the cellular program is performed as a massively parallel computation in which implicit communication occurs only among neighbor cells that access each other state.

2.2 Cellular Lattice Declaration

As mentioned in the previous section, the lattice declaration defines the lattice dimension and the lattice size. Most languages support two-dimensional rectangular lattices only (e.g., CANL and CDL). However, some of them, such as CARPET and Cellang, allow the definition of 1-D, 2-D, and 3-D lattices. Some languages allow also the explicit definition of boundary conditions such as CANL [7] that allows *adiabatic* boundary conditions where absent neighbor cells are assumed to have the same state as the center cell. Others implement *reflecting* conditions that are based on mirroring the lattice at its borders. Most languages use standard boundary conditions such as *fixed* and *toroidal* conditions.

2.3 Cell State

The cell state contains the values of data on which the cellular program works. Thus the global state of an automaton is defined by the collection of the state values of all the cells. While low-level implementations of CA allow to define the cell state as a small number of bits (typically 8 or 16 bits), cellular languages such as CARPET, CANL, DEVS-C++ and CDL allows a user to define cell states as a record of typed variables as follows:

```
cell = (direction :int ;
        speed     : float);
```

where two substates are declared for the cell state. According to this approach, the cell state can be composed of a set of sub-states that are of *integer*, *real*, *char* or *boolean* type and in some case (e.g., CARPET) arrays of those basic types can also be used. Together with the constructs for cell state definition, CA languages define statements for state addressing and updating that address the sub-states by using their identifiers `cell.direction`.

2.4 Neighborhood

An important feature of CA languages that differentiate them from array-based languages and standard data-parallel languages is that that they do not use explicit array indexing. Thus, cells are addressed with a name or the name of the cells belonging to the neighborhood. In fact, the neighborhood concept is used in the CA setting to define interaction among cells in the lattice. In CA languages the neighborhood defines the set of cells whose state can be used in the evolution rule of the central cell. For example, if we use a simple neighborhood composed of four cells we can declare it as follows

```
neigh cross = (up, down, left, right);
```

and address the neighbor cell states by the ids used in the above declaration (e.g., `down.speed, left.direction`). The neighborhood abstraction is used to define the communication pattern among cells. It means that at each tipe step,

a cell send to and receive from the neighbor cells the state values. In this way implicit communication and synchronization are realized in cellular computing. The neighbor mechanism is a concept similar to the *region* construct that is used in the ZPL language [2] where regions replace explicit array indexing making the programming of vector- or matrix-based computations simpler and more concise. Furthermore, this way of addressing the lattice elements (cells) does not require compile-time sophisticated analysis and complex run-time checks to detect communication patterns among elements.

2.5 Parallelism Exploitation

CA languages do not provide statements to express parallelism at the language level. It turns out that a user does not need to specify what portion of code must be executed in parallel. In fact, in parallel CA languages the unit of parallelism is a single cell and parallelism, like communication and synchronization, is implicit. This means that in principle the transaction function of every cell is executed in parallel with the transaction functions of the other cells. In practice, when coarse grained parallel machines are used, the number of cells N is greater than the number of available processors P, so each processor executes a block of N/P cells that can be assigned to it using a domain decomposition approach.

2.6 CA Mapping

Like parallelism and communication, also data partitioning and process-to-processor mapping is implicit in CA languages. The mapping of cells (or blocks of them) onto the physical processors that compose a parallel machine is generally done by the run-time system of each particular language and the user usually intervenes in selecting the number of processors or some other simple parameter. Some systems that run on MIMD computers use load balancing techniques that assign at run-time the execution of cell transition functions to processors that are unloaded or use greedy mapping techniques that avoid some processor to become unloaded or free during the CA execution for a long period. Example of these techniques can be found in [15], [6] and [1].

2.7 Output Visualization and Monitoring

A computational science application is not just an algorithm. Therefore it not sufficient to have a programming paradigm for implementing a complete application. It is also as much significant to dispose of environments and tools that help a user in all the phases of the application development and execution. Most of the CA languages we are discussing here provide a development environment that allows a user not only to edit and compile the CA programs. They allow to monitor the program behavior during its execution on a parallel machine, by visualizing the output as composed of the states of all cells. This is done by displaying the numerical values or by associating colors to those values. Examples

of these parallel environments are CAMEL for CARPET, PECANS for CANL, and DEVS for DEVS-C++. Some of these environments provide dynamical visualization of simulations together with monitoring and tuning facilities. Users can interact with the CA environment to change values of cell states, simulation parameters and output visualization features. These facilities are very helpful in the development of complex scientific applications and make possible to use those CA environments as real problem solving environments (PSEs) [4].

Here we addressed the most important aspects that concern the CA software development process from problem specification to execution and simulation tuning. In the next sections we use the CARPET language as a case-study language to describe in practice how cellular languages can support the development of computational science applications.

3 CARPET: A high-level cellular language

CARPET implements the main CA features in a high-level programming language to assists parallel cellular algorithms design without apparent parallelism [10]. In particular, CARPET has been used for programming cellular algorithms in the CAMEL (Cellular Automata environMent for systEms ModeLing) parallel environment [1]. CAMEL provide a software environment designed to support the parallel execution of cellular algorithms, the visualization of the results, and the monitoring of the program execution. CARPET and CAMEL have been used for implementing high-performance simulations of lava flows, landslides, freeway traffic, and soil bioremediation [11].

The execution of cellular algorithms is implemented by the parallel execution of the transition function of every cell according to the Single Program Multiple Data (SPMD) model. In this way CAMEL exploits the computing power of a highly parallel computer, hiding the architecture issues from a user. A CARPET user can design cellular programs describing the actions of many simple active elements (implemented by the cells) interacting locally. Then, the CAMEL system allows a user to observe the global complex evolution that arises from all the local interactions.

According to the SPMD programming approach, a user must define by CARPET the transition function of a single cell of the system he wants to simulate, then the language run-time system executes transition function in parallel to update the state of each cell. The main features of CARPET are the possibility to describe the state of a cell as a record of typed substates each one by a user-defined type, and the simple definition of complex neighborhoods (e.g., hexagonal) that can be also time dependent in a n-dimensional discrete Cartesian space.

By CARPET, a variety of cellular algorithms can be designed in a simple but very expressive way. The language utilizes the control structures, the types, the operators and the expressions of the C language and it enhances the declaration part allowing the declaration of the features of a cellular automaton. These

are the dimensions of the automaton (e.g., the declaration **dimension 3;** defines a three dimensional automaton), the radius (**radius**) of the neighborhood and the pattern of the neighborhood (**neighbor**). For example, a very simple neighborhood composed of four cells can be defined as follows:

```
neighbor Stencil[4] ([-1,0]Left, [1,0]Right, [0,1]Up, [0,-1]Down);
```

As mentioned before, the state (**state**) of a cell is defined as a set of typed substates that can be *shorts, integers, floats, char,* and *doubles* or *arrays* of these basic types. In the following example, the state consists of three substates.

```
state (float speedX, speedY, energy);
```

The *energy* substate of the current cell can be referenced by the predefined variable **cell_energy**. The neighbor declaration assigns a name to specified neighboring cells of the current cell and allows such to refer to the value of the substates of these identified cells by their name (e.g., **Left_energy**). Furthermore, the name of a vector whose length is the number of elements composing the logic neighborhood it must be associated to the neighborhood (e.g., **Stencil**). The name of the vector can be used as an alias in referring to the neighbor cells. Through the vector, a substate can be referred as **Stencil[i]_energy** where $0 \leq i < 4$.

To guarantee the semantics of cell updating in cellular automata the value of one substate of a cell can be modified only by the **update** operation, for example

```
update(cell_speedX, 13.4);
```

After the execution of an **update** statement, the value of a substate argument remains unchanged in the current iteration. The new value takes effect at the beginning of next iteration. Furthermore, a set of global parameters (**parameter**) can be declared to define global characteristics of the system to be simulated (e.g., the permeability of a soil). Finally, CARPET allows users to define cells with different transition functions (inhomogeneous CA) by means of the **GetX, GetY, GetZ** functions that return the value of the coordinate X, Y, and Z of the cell in the automaton. By varying only a coordinate it is possible, for example, to associate the same transition function to all cells belonging to a plane in a three dimensional automaton.

The language does not provide statements to configure the automata, to visualize the cell values or to define data channels that can connect the cells according to different topologies. The configuration of a cellular automaton is defined by the graphical user interface (UI) of the CAMEL environment. The UI allows, by menu pops, to define the size of the cellular automata, the number of the processors onto which the automata must be executed, and to choose the colors to be assigned to the cell substates to support the graphical visualization of their values. The exclusion of constructs for configuration and data visualization from the language it allows to execute the same CARPET program using different configurations. Furthermore, it makes possible to change from time to time the size of the automaton and/or the number of the processors onto which

the automaton must be executed. Finally, this approach allows selecting the more suitable range of the colors for visualization of data.

4 Practical examples of cellular programming

In this section we describes two practical examples of cellular programming written using the CARPET language. The first example is a typical CA application that simulates excitable systems. The second program is the classical Jacobi relaxation that shows how it is possible to use CA languages not only for simulate complex systems and artificial life models, but that they can be used to implement parallel programs in the area of fine grained applications such as finite elements methods, partial differential equations and systolic algorithms that are traditionally developed using array or data-parallel languages.

4.1 The Greenberg-Hastings model

A classical model of excitable media was introduced 1978 by Greenberg and Hastings [5]. This model considers a two-dimensional square grid. The cells are in one of a *resting* (0), *refractory* (1), or *excited* (2) state. Neighbors are the eight nearest cells. A cell in the resting state with at least s excited neighbors (in the program we use $s = 1$) becomes excited itself, runs through all excited and resting states and returns finally to the resting state. A resting cell with less than s excited neighbors stays in the resting state.

Excitable media appear in several different situations. One example is nerve or muscle tissue, which can be in a resting state or in an excited state followed by a refractory (or recovering) state. This sequence appears for example in the heart muscle, where a wave of excitation travels through the heart at each heartbeat. Another example is a forest fire or an epidemic model where one looks at the cells as infectious, immune, or susceptible.

Figure 1 shows the CARPET program that implements the two-dimensional Greenberg-Hastings model. It appears concise and simple because the programming level is very close to the model specification. If a Fortran+MPI or C+MPI solution is adopted the source code is extremely longer with respect to this one and, although it might be a little more efficient, it is very difficult to program, read and debug.

4.2 The Jacobi relaxation

As a second example, we describe the four-point Jacobi relaxation on a $n \times n$ lattice in which the value of each element is to be replaced by the average value of its four neighbor elements. The Jacobi relaxation is an iterative algorithm that is used to solve differential equation systems. It can be used, for example, to compute the heat transfer in a metallic plate on which boundaries there is a given temperature. At each step of the relaxation the heat of each plate point (cell) is updated by computing the average of its four nearest neighbor points.

```
#define resting 0
#define refractory 1
#define excited 2

cadef
{
  dimension 2;
  radius 1;
  state (short value);
  neighbor Moore[8] ([0,-1]North, [1,-1]NorthEast,[1,0]East,
                     [1,1]SouthEast,[0,1]South,[-1,1]SouthWest,
                     [-1,0]West, [-1,-1]NorthWest);
}
 int i, exc_neigh=0;
{
 for (i=0; (i<8) && (exc_neigh==0); i++)
   if (Moore[i]_value == excited) exc_neigh = 1;
 switch (cell_value)
 {
   case excited    : update(cell_value, recovering); break;
   case recovering : update(cell_value, resting); break;
   default         : /* cell is in the resting state */
                     if (exc_neigh == 1)
                       update(cell_value, excited);
 }
}
```

Fig. 1. The Greenberg-Hastings model written in CARPET.

Figure 2 shows a CARPET implementation. The initial `if` statement is used to set the initial values of cells that are taken to be 0.0 except for the western edge where boundary values are 1.0.

The Jacobi program, although it is a simple algorithm, is another example of how a CA language can be effectively used to implement scientific programs that are not properly in the original area of cellular automata. This simple case illustrates the high-level features of the CA languages that can be also used for implement applications that are based on the manipulation of arrays such as systolic algorithms and finite elements methods.

For the Jacobi algorithm we present some performance benchmarks that have been obtained by executing the CARPET program using different grid sizes and processor numbers. Table 1 shows the execution times for 100 relaxation steps for three different grid sizes (100×200, 200×200 and 200×400) on 1, 2, 4, 8 and 10 processors of a QSW CS-2 multicomputer. From the figure we can see that as the number of used processors increases, there is a corresponding decrease of the execution time. This trend is more evident when larger grids are used; while smaller CA do not use efficiently the processors. This means that, because

```
cadef
{
 dimension 2;
 radius 1;
 state ( float elem );
 neighbor Neum[4]([0,-1]North,[-1,0]West,[0,1]South,[1,0]East);
}
   int sum;
{
 if (step == 1 )
   if (GetY == 1)
     update (cell_elem, 1.0);
   else
     update (cell_elem, 0.0);
 else
   {
    sum = North_elem+South_elem+East_elem+West_elem;
    update (cell_elem, sum/4);
   }
}
```

Fig. 2. The Jacobi iteration program written in CARPET.

of the algorithm simplicity, when we run an automaton with a small number of cells we do not need to use several processing elements. On the contrary, when the number of cells in the lattice is high, the algorithm benefits from the use of a higher number of computing resources. This can be also deduced from table 2 that shows the relative speed up results for the three different grids. In particular, we can observe that when a 200×400 lattice of cells is used we obtain a superlinear speed up in comparison to the sequential execution mainly because of memory allocation and management problems that occur when all the 80,000 cells are allocated on one single processing element.

Table 1. Execution time (in sec.) of 100 iterations for the Jacobi algorithm

Grid Sizes	1 Proc	2 Procs	4 Procs	8 Procs	10 Procs
100×200	1.21	0.65	0.37		0.25
200×200	3.62	1.25	0.67	0.42	0.37
200×400	8.22	3.65	1.26	0.74	0.62

Table 2. Relative speed up of the Jacobi algorithm

Grid Sizes	1 Proc	2 Procs	4 Procs	8 Procs	10 Procs
100×200	1	1.86	3.27		4.84
200×200	1	2.89	5.40	8.62	9.78
200×400	1	2.25	6.52	11.10	13.25

5 Conclusions

The primary function of programming languages and tools has always been to make the programmer more effective. Appropriate programming languages and tools may drastically reduce the costs for building new applications as well as for maintaining existing ones. It is well known that programming languages can greatly increase programmer productivity by allowing the programmer to write high-scalable, generic, readable and maintainable code. Also, new domain specific languages, such as CA languages, can be used to enhance different aspects of software engineering. The development of these languages is itself a significant software engineering task, requiring a considerable investment of time and resources. Domain-specific languages have been used in various domains and the outcomes have clearly illustrated the advantages of domain specific-languages over general purpose languages in areas such as productivity, reliability, and flexibility.

The main goal of the paper is answering the following question: How does one program cellular automata on parallel computers? We think that it is very important for an effective use of cellular automata for computational science on parallel architectures to develop and use high-level programming languages and tools that are based on the cellular computation paradigm. These languages may provide a powerful instrument for scientists and engineers that need to implement real-life applications on parallel machines using a fine-grain approach. This approach allows designers to concentrate on "how to model a problem" rather than on architectural details as occurs when people use low-level languages that have not been specifically designed to express fine-grained parallel cellular computations.

In a sense, parallel cellular languages provide a *high-level paradigm* for fine-grain computer modeling and simulation. While efforts in sequential computer languages design focused on *how* to express sequential objects and operations, here the focus is on finding out *what* parallel cellular objects and operations are the ones we should want to define [9]. Parallel cellular programming is emerging as a response to these needs.

After discussing the main issues in programming scientific applications by means of parallel cellular languages, we discussed the CARPET language as an example in this class of languages. By CARPET we described the implemention of two application examples that illustrate the main features of this approach.

Currently CARPET and the latest version of its run-time system named CAMELot (*CAMEL open technology*) are used for the implementation of models and simulation of complex phenomena and they are available on parallel architectures and cluster computing systems that use Sun Solaris, SGI IRIX, Red Hat Linux and Tru64 UNIX 4.0F as operating systems.

Acknowledgements

This research has been partially funded by the CEC ESPRIT project n 24,907.

References

1. Cannataro, M., Di Gregorio, S., Rongo, R., Spataro W., Spezzano, G., Talia, D.: A Parallel Cellular Automata Environment on Multicomputers for Computational Science. *Parallel Computing* 21 (1995) 803–824.
2. Chamberlain, B.L., et al.: The Case for High-Level Parallel Programming in ZPL. *IEEE Computational Science and Engineering*. 5 (1998) 76–86.
3. Eckart, J.D.: Cellang 2.0: Reference Manual, *ACM Sigplan Notices*. 27 (1992) 107–112.
4. Gallopoulos, E., Houstis, E.N., Rice, J.R.: Workshop on Problem-Solving Environments: Findings and Recommendations. *ACM Computing Surveys* 27 (1995) 277-279.
5. Greenberg, J. M., Hastings, S.P.: Spatial Patterns for Discrete Models of Diffusion in Excitable Media. *SIAM J. Appl. Math.* 34 (1978) 515–523.
6. Hochberger, C., Hoffmann, R.: CDL - A Language for Cellular Processing. *Proc. 2nd Int. Conf. on Massively Parallel Computing Systems*. IEEE Computer Society Press (1996).
7. Mango Furnari, M., Napolitano, R.: A Parallel Environment for Cellular Automata Network Simulation. *Proc. 2nd International Workshop on Massive Parallelism*. World Scientific, Singapore (1994) 353-364.
8. Seutter, F.: CEPROL: A Cellular Programming Language, *Parallel Computing* 2 (1985) 327–333.
9. Sipper, M.: The Emergence of Cellular Computing. *IEEE Computer* 32 (1999) 18–26.
10. Spezzano G., Talia, D.: A High-Level Cellular Programming Model for Massively Parallel Processing. *Proc. 2nd Int. Workshop on High-Level Programming Models and Supportive Environments (HIPS97)*. IEEE Computer Society Press (1997) 55–63.
11. Spezzano G., Talia, D.: Designing Parallel Models of Soil Contamination by the CARPET Language. *Future Generation Computer Systems* 13 (1998) 291–302.
12. Talia, D., Sloot, P.: Cellular Automata: Promise and Prospects in Computational Science. *Future Generation Computer Systems* 16 (1999) v–vii.
13. Vialle, S., Lallement, Y., Cornu, T.: Design and Implementation of a Parallel Cellular Language for MIMD Architectures. *Computer Languages* 24 (1998) 125–153.
14. von Neumann, J.: *Theory of Self Reproducing Automata*. University of Illinois Press (1966).
15. Weimar, J.R.: *Simulation with Cellular Automata*, Logos-Verlag, Berlin (1997).

16. Wolfram, S.: Computation Theory of Cellular Automata, *Comm. Math. Phys.* 96 (1984).
17. Worsch, T.: Programming Environments for Cellular Automata. *Cellular Automata for Research and Industry (ACRI 96)*. Springer-Verlag, London (1996) 3–12.
18. Zeigler, P.Z., et al.: The DEVS Environment for High-Performance Modeling and Simulation. *IEEE Computational Science & Engineering* (1997) 61–71.

A Novel Algorithm for the Numerical Simulation of Collision-Free Plasma-Vlasov Hybrid Simulation

David Nunn[1]

[1] Department of Electronics and Computer Science,
Southampton University,
Southampton, Hants, SO17 1BJ, UK.

Abstract. Numerical simulation of collision-free plasma is of great importance in the fields of space physics, solar and radio physics, and in confined plasmas used in nuclear fusion. This work describes a novel completely general and highly efficient algorithm for the numerical simulation of collision-free plasma. The algorithm is termed Vlasov Hybrid Simulation (VHS) and uses simulation particles to construct particle distribution function in the region of phase **(r,v)** space of interest. The algorithm is extremely efficient and far superior to the classic particle in cell method. A fully vectorised and parallelised VHS code has been developed, and has been successfully applied to the problem of the generation of VLF triggered emissions and VLF 'dawn chorus', due to the nonlinear interaction of cyclotron resonant electrons with narrow band VLF band waves (~kHz) in the earth's magnetosphere.

1 Introduction

The problem of the numerical simulation of plasma is one of great importance in the realms of both science and engineering. The physics of the solar corona is essentially that of a very hot collision free plasma. Plasma physics governs the behaviour of radio waves in the whole of the earth's near space region, usually termed the 'magnetosphere'. Closer to home plasmas employed in nuclear fusion devices and industrial plasmas may well have time and spatial scales which make them effectively collision-free, and understanding their dynamics is of vital importance.

The equations governing any collision free (CF) plasma physics problem are those of Maxwell and Liouville. Liouville's theorem states that the density of particles F(**r**,**v**) in 6 dimensional *phase space* ***r,v*** is conserved following the trajectories of particles in phase space. Clearly plasma physics problems may be immensely complicated, particularly if

particle motion is non linear. Usually one must resort to numerical simulation to gain any comprehension of what is happening.

Traditionally the methodology of choice for Collision Free plasma simulation was the classic particle-in-cell (PIC) method. The required spatial domain r or simulation **box** is covered by a suitable grid. A large number of simulation particles (SP's) are inserted into the simulation box and their trajectories followed according to the usual equations of motion. At each time step particle charge/currents are assigned or distributed to the immediately adjacent spatial grid points, thus giving the charge/current field in the box. Use of the discretised Maxwell's equations allows one to time advance or 'push' the electric and magnetic field vectors in the **r** domain. PIC codes however suffer from several disadvantages. They are noisy, make inefficient use of simulation particles, and do not properly resolve distribution function in phase space. For problems involving small amplitude waves where the perturbation in distribution function dF is relatively small (dF<<Fo) they are particularly noisy and inefficient.

2 The Vlasov Hybrid Simulation Method (VHS)

A novel and highly efficient simulation method has been devised termed Vlasov Hybrid Simulation (VHS) [1]. The structure of the algorithm is as follows. A phase space **(r,v)** simulation box is first selected, to cover the domain of interest in the problem at hand. The maximum dimensionality of phase space is 6, but many realistic simulations have a reduced number of spatial or velocity space dimensions. The phase box may be a function of time as the simulation progresses. In the present case for example we are interested in electrons that are cyclotron resonant with the wave field and this phase box will cover the region of velocity space that is close to the resonance velocity. The box is filled with a grid to provide adequate resolution of distribution function in phase space. At the start of the simulation the phase box is evenly filled with particles at a density of about 1-2 per elementary grid cell. By Liouville's theorem distribution function F is conserved along phase trajectories. Each Simulation Particle (SP) is assigned a value of F appropriate to the initial conditions for the problem at hand. As the simulation progresses the SP trajectories in phase space are numerically integrated, in this case using a second order modified Euler algorithm. Thus the value of distribution function (F) is known at the points in phase space where the simulation particles happen to be located. Now at each time step the values of F at SP points are *interpolated* to the fixed phase space grid. This is achievable by a very simple procedure. The value of distribution function F^l at each Simulation Particle number l is distributed additively to adjacent grid points using the familiar area weighting coefficients α_l as employed in classical PIC codes. The weighting coefficients α_l themselves are also distributed additively to adjacent grid points. For a specific grid point ijk we then have

$$F_{ijk} = \left[\sum_l F^l \alpha_l\right] / \left[\sum_l \alpha_l\right]$$

where the sum is over all simulation particles located in the 2^n elementary hypercubes surrounding the grid point in question, where a phase space of dimensionality n has been assumed. This interpolation procedure lies at the heart of the VHS method and confers its many advantages. Once distribution function F_{ijk} is defined on a regular velocity space grid it is a simple matter to compute plasma current and charge fields in 3D cartesian space, by appropriate integration (summation) over the velocity space grid. Following this one may push the EM fields forwards in time using a discretised representation of Maxwell's equations.

2.1 Particle control

Fortunately from Liouville's theorem itself there is no tendency for SP's to bunch and leave grid points 'uncovered'. Where this does occur a value for F_{ijk} may be secured by interpolation from neighbouring grid points. At any time extra SP's may be inserted into (or removed from) the phase fluid-they only act as markers providing information about the value of distribution at a particular point. Unlike all other techniques the density of SP's is not a critical quantity. It only needs to remain at a value greater than ~1 per elementary phase space volume. For some problems, and this is particularly true in the present case, there will be a flux of phase fluid out of or into the simulation phase space box along its boundaries. Particles leaving the phase box are discarded as they convey information no longer required. Where phase fluid enters the box it is necessary to insert new SP's into the phase fluid at that point. This has to be done with some care in order to attain an acceptable density of simulation particles in the incoming phase fluid. It is the interpolation procedure that makes it legitimate and possible to do this. This is a very powerful feature of VHS. The population of simulation particles is dynamic and constantly changing.

2.2 Advantages of VHS

VHS has been found to be highly efficient and to have very low noise levels when compared to PIC codes. Very efficient use is made of the simulation particles, as they carry information as to the value of F (or rather dF). Unlike other Vlasov simulation techniques that have been developed the algorithm is very stable and robust. For example the standard method of Cheng and Knorr [2] aims to solve numerically the Vlasov equations in phase or configuration space. This requires the determination of the gradient of distribution function in phase space. This presents severe practical problems. In many plasma simulation problems particle distribution function acquires quite legitimately fine structure in phase space, often termed 'filamentation'. For example this may arise in wave

particle interaction problems in plasma when particles become phase trapped in a narrow band wave. Such filamentation makes the Cheng and Knorr algorithm numerically unstable against filamentation in velocity space. Attempts to resolve this problem involve techniques such as numerical smoothing, which corrupts the underlying physics being simulated. Another feature of VHS is that for certain problems,one may limit the region of phase space where F is resolved to a time varying simulation box.This is indeed the case in the present problem. The ability to accommodate a flux of phase fluid across the boundary of the phase box is unique to VHS and allows the particle population to be dynamic and to change constantly. In this way the particle population is confined to a set that is locally optimal in time. For example in the present problem particles are constantly drifting into and out of resonance with the wave. A PIC code would end up following large numbers of non resonant particles, but a VHS code will constantly discard non resonant particles and continually introduce new resonant particles. The benefits in computational time this confers cannot be over estimated.

Another virtue of the VHS method is that distribution function is properly resolved in phase space and is available as a diagnostic output. Distribution function is only available from a PIC code by numerically inspecting the density of (weighted) simulation particles in velocity space. This is actually rarely done with PIC codes, and if it were one would quickly realise that the density of SP's was grossly inadequate to define F, let alone dF. It is a fact that PIC codes, particularly in applications with high dimensionality, often have inadequate numbers of simulation particles. The noise level is then extremely high, and the authors are relying on integration over time and over velocity space (in the evaluation of $J(r)$ and $\rho(r)$) to reduce the noise to manageable levels.

3 The application area.

The VHS algorithm has been fruitfully applied here to a classic problem in space plasma physics. This is the generation mechanism of triggered emissions and chorus in the VLF band (3-30kHz) in the earth's magnetosphere. Triggered emissions are narrow band signals with sweeping frequency. Typically the frequency may rise or fall by several kHz in a time ~1-2 secs. More complex spectral forms are often observed, such as downward hooks, upwards hooks, quasi constant tones and emissions whose frequency oscillates. Emissions are generated as a result of nonlinear cyclotron resonant interaction between the the EM wave and energetic radiation belt electrons of ~keV energy. Emissions achieve quite strong amplitudes of $B \simeq 2\text{-}10pT$, which represents a wave strong enough to nonlinearly 'trap' cyclotron resonant electrons. It is generally agreed that chorus and VLF emission arise in 'ducts' where the wave vector is closely parallel to the ambient magnetic field direction. A key aspect of the nonlinear wave particle interaction is the dominant role of the magnetic field inhomogeneity, which controls particle trapping dynamics and confines the interaction region to the equatorial zone. Consequently we have developed a

VHS/VLF code with 1 spatial dimension and 3 velocity dimensions to simulate *this nonlinear self consistent* interaction in the equatorial zone of the earth's magnetosphere. The region of generation is typically between 3 and 10 earth radii in altitude and some 1000s of kms in extent, spread along equatorial magnetic field lines. This problem is extremely well suited to the VHS method-indeed this simulation has not been successfully achieved with any other type of simulation method, and PIC codes have shown themselves to be quite incapable of simulating this phenomenon. The phase box encloses the cyclotron resonance velocity Vres

$V_{res} = (\omega - \Omega)/k$

where ω is wave frequency, Ω is electron gyrofrequency and k is wave number. Note that resonance velocity is in the opposite direction to wave phase and group velocities. The resonance velocity will vary in both space and time, through changing frequency of the emission, and significantly through inhomogeneity of the ambient magnetic field, which has a parabolic dependence on distance z from the equator . Thus particles are constantly entering the phase box, which is the region of resonance and thus of direct physical interest. It is thus guaranteed that all SP's are close to resonance.

4 The VHS/VLF code.

The code has been developed in Fortran77 and has been run on a wide variety of platforms, namely Origin2000, DEC Alpha cluster, Convex Exemplar, Cray YMP etc. The most numerically intensive procedures are the particle push routines, and the process of interpolating distribution function from particles to the fixed grid. The particle push routines fully vectorize, but the interpolation procedure does not due to its logical complexity. The whole code has been parallelised using MPI, which has been easily achieved by means of the following technique. The 1D spatial domain is divided into M adjacent blocks, where M is the number of available processors. Each processor implements the particle push and interpolation procedures in its part of the spatial grid. At each step, those particles which physically move from one spatial domain to the next must be passed with their appertaining data between adjacent processors at the interface. The field push equations and certain global operations such as FFT/IFFT filtering of the EM wave fields are low work load operations and are performed by the master processor. All processors must pass current field data to the master at each timestep, where field push and field filtering are performed. The master then returns the new global EM wave fields to the 'slaves' who then perform the particle push and distribution function interpolation for the next timestep.

The simulation takes place within a finite frequency band located about a centre frequency which itself may be a function of time. The simulation bandwidth is ~ 70Hz which requires a spatial grid ~1600 in order to resolve all Fourier components of the wave spectrum. The velocity space grid must be dense enough to resolve the structure of the distribution function in the region about the resonance velocity. The dominant structure is the so called 'resonant particle trap' and it was found that having 50 grid points in the Vz axis parallel to the Bo direction and 20 points in gyrophase gave adequate resolution. The total number of phase space grid points and thus the number of simulation particles is thus typically in the range 0.5-5 million. A short run may take only a few hours on an Origin2000. However run time scales as bandwidth cubed, so high bandwidth runs may take as long as a week.

5. The observational data

Radio emissions in the VLF (kHz) band, the so called VLF emissions, may occur spontaneously or be obviously triggered by some other signal. The first observations of triggered VLF emissions were obtained on the earth's surface on US Navy vessels. Morse code signals at 14kHz from the high power VLF transmitter NAA at Cutler, Maine were observed to 'trigger' long enduring radio emissions (~1 second) with a sweeping frequency ~2kHz/sec. [3]. In pioneering research it was realised by Helliwell [3] that these emissions must arise in the earth's magnetosphere and be due to non linear electron cyclotron resonance with radiation belt electrons with energies ~keV. Since that time triggered VLF emissions have been routinely observed on the ground, particularly at Halley Bay,Antarctica [4] and in Northern Scandinavia [5]. In the 1970's Stanford University established a horizontal VLF antenna in Antarctica at Siple station on the South Polar plateau.[6]. An extensive program of VLF transmissions were made to probe the magnetosphere and investigate the phenomenon of triggered emissions. One of the main objectives of the research program described here has been to develop the theory and numerical simulation tools to fully understand the many extraordinary phenomena observed in the Siple data base.

Since triggered emissions are generated in space, it is not surprising that this phenomenon has also been observed on board scientific satellites. Unfortunately the VLF radio waves are confined to field aligned ducts caused by localised enhancements of plasma density. These ducts are ~100km in extent and it is only infrequently that a satellite will pass through a duct. Consequently satellite observations can be rather disappointing. However at large distances from the earth ,~10 earth radii, VLF signals are not ducted and satellites there record a variety of VLF chorus and triggered emissions. A recent paper by Nunn et al (1997) [7] presents VLF emission observations from the Geotail satellite and uses the VHS simulation code to produce almost exact replicas of emissions observed, using all the field and particle observations from on board the satellite. These results confirmed totally the plasma theory underlying this phenomenon.

6. Numerical modelling of Siple triggered emissions

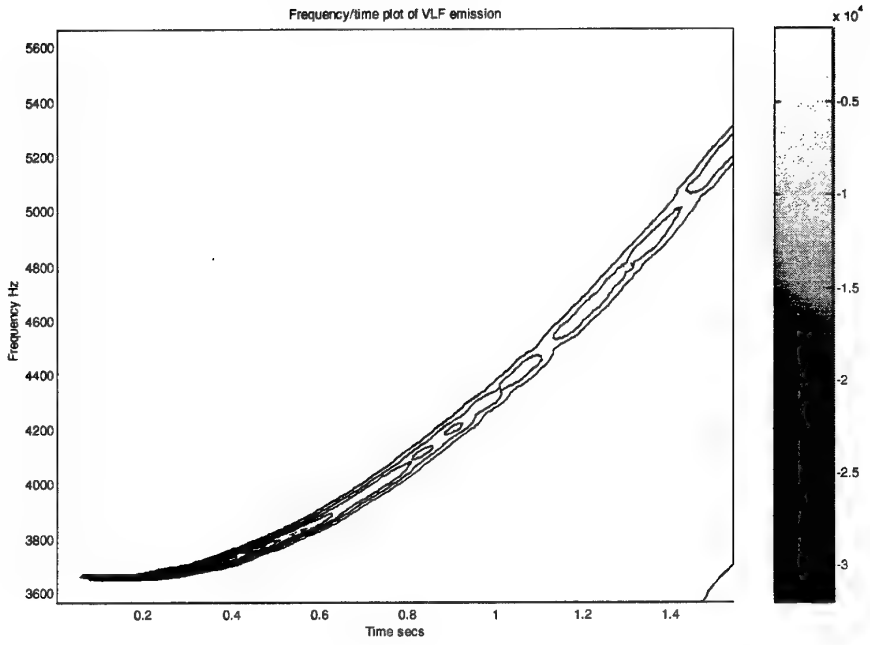


Fig 1.

The VHS/VLF code has been used to successfully simulate the triggering of a rising frequency emission triggered by a CW 70ms pulse at 3663Hz from the Siple transmitter. Figure 1 above displays a frequency-time contour plot of the output wavefield sequence as recorded at the end of the simulation box. The sweep rate of 1kHz/s is in excellent agreement with observations on the ground and on board satellites.

The emission itself is produced by a quasi static non linear self consistent and self maintaining structure termed a VLF soliton or generating region. This soliton is stable in nature, both in reality and in the simulation code. The profile of the riser soliton is shown

in figure 1. The code has completely elucidated the dynamical structure of the VLF soliton, and identified two distinct types, one associated with a riser and one with a faller.

The code is also able to reproduce fallers with a suitable choice of initial parameters. Both downward and upward hooks may be produced by the code and these are interpretable in terms of transitions between the two soliton types. The sweeping frequency is due to the out of phase component of resonant particle current that sets up spatial gradients of wave number in the wave field and is able to sustain these. The top panel of figure 2 shows d/dz(Ji/|R|), where Ji is the out of phase component of resonant particle current and R the complex field. This quantity is the 'driver' that sets up the appropriate wave number gradients.

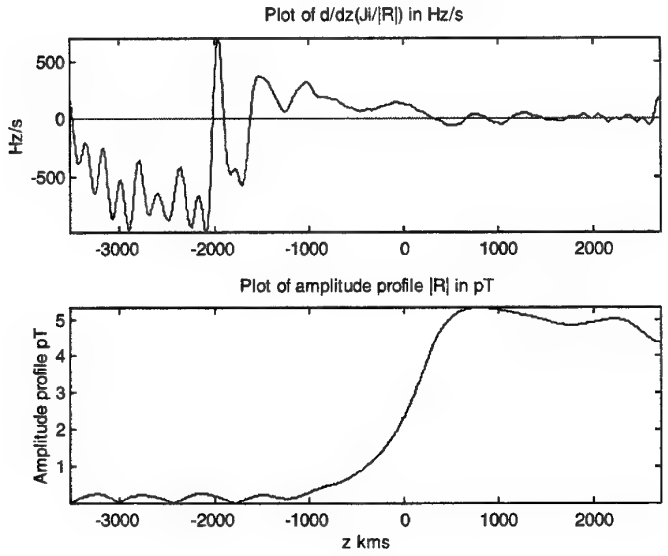


Fig 2.

Figure 3 shows the magnitude of the exit field received at the downstream end of the simulation box. Received amplitude rises rapidly to reach the saturation level, and remains there for a self sustaining emission.

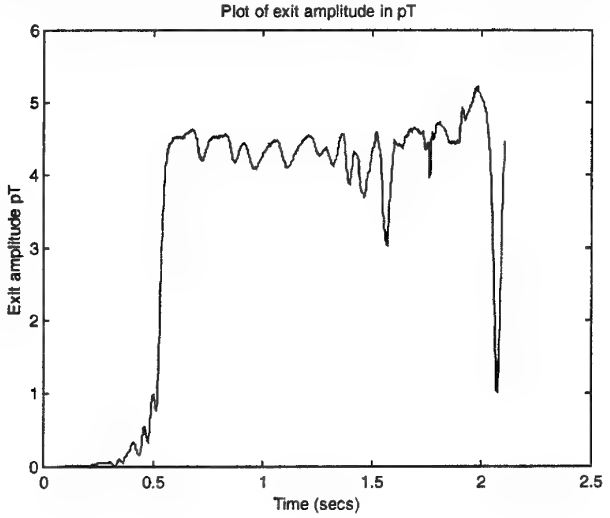


Fig 3

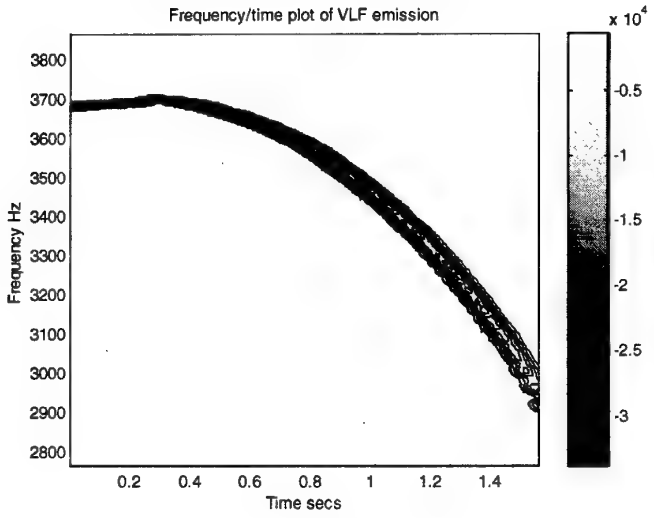


Fig 4

Figure 4 shows the simulation of a falling tone, which was obtained by increasing the ambient linear growth rate to 120dB/s, which has the effect of driving the wave profile upstream and turns the generation region into the structure of the faller type. Again the sweep rate of -1kHz/s is in excellent agreement with observations. For this case fig 5 shows a time snapshot of the wave profile |R| and also of the wave number shift driver d/dz(Ji/|R|). It is seen that the profile now extends further upstream.

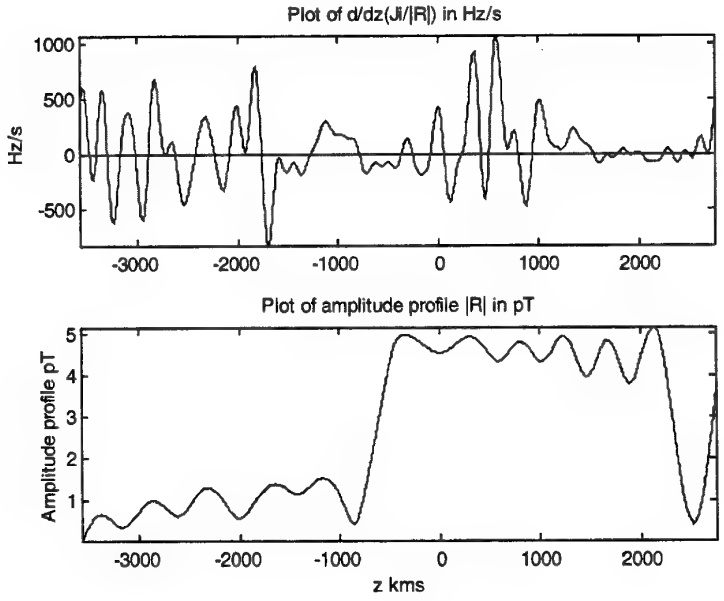


Fig 5

7 Conclusions

The VHS method for numerical simulation of collision free plasma is low noise, highly efficient, very stable and provides excellent diagnostics. In this application the method has been successfully used to simulate triggered radio emissions in the VLF band in the earth's near space region. This is a complex and difficult problem which has never been solved using PIC codes. The VHS method far outperforms particle in cell codes in all applications where resolution of the distribution function in velocity space is required. The method is completely general and may be safely applied to ANY collision free plasma simulation problem. Problems with a high dimensionality will be expensive if tackled with a Vlasov VHS code. However use of a properly constituted Vlasov code guarantees accuracy and meaningful results. It is all too easy to run PIC codes with far too few particles, and to obtain results which although often plausible are in fact heavily corrupted by simulation noise.

References

1. Nunn,D. : A Novel Technique for the Numerical Simulation of Hot Collision Free Plasma-Vlasov Hybrid Simulation. J. of Computational Physics, vol 108(1) (1993) 180-196.

2. Cheng,C.Z., Knorr,G.: The Integration of the Vlasov Equation in Configuration Space. J. Geophysical Research, (1976), vol 95 15073 et seq.

3. Helliwell,R.A. : Whistlers and Related Ionospheric Phenomena, Stanford University

Press, Stanford, California,USA,(1965).

4. Smith A.J. and Nunn,D. : A Numerical Simulation of VLF Risers, Fallers and Hooks observed in Antarctica, J. Geophysical Research,103,(1998) 6771-6784.

5. Nunn,D.,Manninen,J., Turunen,T.,Trakhtengerts,V., and Erokhin,N.: On the Nonlinear Triggering of VLF Emissions by Power Line Harmonic Radiation, Annales Geophsicae,17,(1999),79-94.

6. Helliwell,R.A., Controlled Stimulation of VLF Emissions from Siple Station, Antarctica, Radio Science,18,(1983),801-814.

7. Nunn,D.,Omura,Y.,Matsumoto,H.,Nagano,I., and Yagitani,S.: The Numerical Simulation of VLF Chorus and Discrete Emissions Observed on the Geotail Satellite Using a Vlasov Code, J. Geophysical Research,102,A12,(1997),27083-27097.

Parallelization of a Density Functional Program for Monte-Carlo Simulation of Large Molecules

J. M. Pacheco[1] and José Luís Martins[2]

[1] Departamento de Física da Universidade, 3000 Coimbra, Portugal,
pacheco@hydra.ci.uc.pt,
WWW home page: http://aloof.fis.uc.pt/
[2] Departamento de Física, Instituto Superior Técnico
Av. Rovisco Pais, 1049-001 Lisboa, PORTUGAL
and
INESC Rua Alves Redol, 9, 1000-029 Lisboa, Portugal
jlm@plana.inesc.pt,
WWW home page: http://bohr.inesc.pt/~ jlm

Abstract. A first-principles program designed to compute, among other quantum-mechanical observables, the total energy of a given molecule, is efficiently parallelized using MPI as the underlying communication layer. The resulting program fully distributes CPU and memory among the available processes, making it possible to perform large-scale Monte-Carlo Simulated Annealing computations of very large molecules, exceeding the limits usually attainable by similar programs.

1 Introduction

At present, an enormous effort is being dedicated to the study and fabrication of nano-structures and new materials, which calls for a framework to compute, from *first-principles*, and predict, whenever possible, properties associated with these types of systems. Among such frameworks, Density Functional Theory (DFT) constitutes one of the most promising. Indeed, the success of DFT to compute the ground-state of molecular and solid-state systems has been recognized in 1998 with the award of the Nobel Prize of Chemistry to Walter Kohn and John Pople. DFT provides a computational framework with which the properties of molecules and solids can, in certain cases, be predicted within chemical accuracy (= 1 Kcal/mol). Therefore, it is natural to try to use at profit the most recent computational paradigms in order to break new frontiers in these areas of research and development.

In this work we report the successful parallelization of an *ab-initio* DFT program, which makes use of a Gaussian basis-set. This, as will become clear in the following section, is just one of the possible ways one may write down a DFT-code. It has, however, the advantage of allowing the computation of neutral and charged molecules at an equal footing, of making it possible to write the code in a modularized fashion (leading to an almost ideal load-balance), as well as it

is taylor-made to further exploit the recent developments of the so-called order-N techniques. As a result, the program enables us to carry out the structural optimization of large molecules via a Monte-Carlo Simulated Annealing strategy.

Typically, the implementation of a molecular DFT-code using Gaussian, localized, basis-states, scales as N_{at}^3, or N_{at}^4, depending on implementation, where N_{at} is the number of atoms of the molecule. Such a scaling constitutes one of the major bottlenecks for the application of these programs to large (> 50 atoms) molecules, without resorting to dedicated supercomputers. The fact that the present implementation is written in a modular fashion makes it simple and efficient to distribute the load among the available pool of processes. **All** tasks so-distributed are performed locally in each process, and **All** data required to perform such tasks is also made available locally. Furthermore, the distribution of memory among the available processes is also done evenly, in a non-overlapping manner. In this way we optimize the performance of the code both for efficiency in CPU time as well as in **memory** requirements, which allows us to extend the range of applicability of this technique.

This paper is organized as follows: In Section II a brief summary of the underlying theoretical methods and models, as applied to molecules, is presented, in order to set the framework and illustrate the problems to overcome. In Section III the numerical implementation and strategy of parallelization is discussed, whereas in Section IV the results of applying the present program to the structural optimization of large molecules using Simulated Annealing are presented and compared to other available results. Finally, the main conclusions and future prospects are left to Section V.

2 Molecular Simulations with DFT

In the usual Born-Oppenheimer Approximation (BOA) the configuration of a molecule is defined by the positions $\boldsymbol{R}_i$ of all the N_{at} atoms of the molecule and by their respective atomic number (nuclear charge). The energy of the electronic ground state of the molecule is a function $E_{GS}(\boldsymbol{R}_1, \ldots, \boldsymbol{R}_{N_{at}})$ of those nuclear positions. One of the objectives of quantum chemistry is to be able to calculate relevant parts of that function, as the determination of the full function is exceedingly difficult for all except the simplest molecules. In practice one may try to find the equilibrium configuration of the molecule, given by the minimum of E_{GS}, or one may try to do a statistical sampling of the surface at a given temperature T. That statistical sampling can be done by Molecular Dynamics (MD) or by Monte-Carlo (MC) methods. By combining the statistical sampling at a given T with a simulation process in which one begins at a high T and, after equilibrating the molecule, starts reducing the T in small steps, always equilibrating the molecule before changing T, one realizes an efficient algorithm for the global minimization of E_{GS}, the so-called Simulated Annealing Method (SAM).

The calculation of E_{GS} for a single configuration is a difficult task, as it requires the solution of an interacting many-electron quantum problem. In Kohn-

Sham DFT this is accomplished by minimizing a functional of the independent electron orbitals $\psi_i(\boldsymbol{r})$,

$$E_{GS}(\boldsymbol{R}_1,\ldots,\boldsymbol{R}_{N_{at}}) = \min_{\psi_i} E_{KS}(\boldsymbol{R}_1,\ldots,\boldsymbol{R}_{N_{at}};\psi_1,\ldots,\psi_{N_{el}}) \tag{1}$$

where N_{el} is the number of electrons of the molecule, and the minimization is done under the constraint that the orbitals remain orthonormal,

$$\int \psi_i(\boldsymbol{r})\psi_j(\boldsymbol{r})d^3r = \delta_{ij}. \tag{2}$$

The Euler-Lagrange equation associated with the minimization of the Kohn-Sham functional is similar to a one particle Schrodinger equation

$$-\frac{\hbar^2}{2m}\nabla^2\psi_i(\boldsymbol{r}) + v_{\text{eff}}(\boldsymbol{r};\psi_1,\ldots,\psi_n)\psi_i(\boldsymbol{r}) = \epsilon_i\psi_i(\boldsymbol{r}), \tag{3}$$

except for the non-linear dependence of the effective potential v_{eff} on the orbitals. As our objective here is to discuss the numerical implementation of our algorithms, we will not discuss the explicit form of v_{eff} and the many approximations devised for its practical calculation, and just assume one can calculate v_{eff} given the electron wavefunctions $\psi_i(\boldsymbol{r})$. The reader can find the details on how to calculate v_{eff} in excellent reviews, e. g., refs.[1, 2] and references therein.

If one expands the orbitals in a finite basis-set,

$$\psi_i(\boldsymbol{r}) = \sum_j^M c_{ij}\phi_j(\boldsymbol{r}) \tag{4}$$

then our problem is reduced to the minimization of a function of the coefficients,

$$E_{GS}(\boldsymbol{R}_1,\ldots,\boldsymbol{R}_{N_{at}}) \approx \min_{c_{ij}} E_{KS}(\boldsymbol{R}_1,\ldots,\boldsymbol{R}_{N_{at}};c_{11},\ldots,c_{N_{el}M}) \tag{5}$$

and the Euler-Lagrange equation becomes a matrix equation of the form

$$\sum_j c_{ij}[H_{kj} - \epsilon_i S_{kj}] = 0 \tag{6}$$

where the eigenvalues are obtained, as usual, by solving the secular equation

$$det|H_{ij} - ES_{ij}| = 0. \tag{7}$$

The choice of the basis-set is not unique[3]. One of the most popular basis-sets uses Gaussian basis-functions

$$\phi_i(\boldsymbol{r}) = N_i \exp(-\alpha_i(\boldsymbol{r} - \boldsymbol{R}_i)^2)Z_{l(i)}^{m(i)}(\boldsymbol{r} - \boldsymbol{R}_i) \tag{8}$$

where the angular funtions Z_l^m are chosen to be real solid harmonics, and N_i are normalization factors. These functions are centered in a nucleus $\boldsymbol{R}_i$ and are an

example of localized basis-sets. This is an important aspect of the method, since this implies that the matrix-elements H_{ij} result, each of them, from the contribution of a large summation of three-dimensional integrals involving basis-functions centered at different points in space. This multicenter topology involved in the computation of H_{ij} ultimately determines the scaling of the program as a function of N_{at}. Finally, one should note that, for the computation of H_{ij} one needs to know v_{eff} which in turn requires knowledge of $\psi_i(\boldsymbol{r})$. As usual the solution is obtained via a self-consistent iterative scheme, as illustrated in fig.1 .

Due to the computational costs of calculating E_{GS} from first principles, for a long time the statistical sampling of E_{GS} has been restricted to empirical or simplified representations of that function. In a seminal paper, Car and Parrinello[4] (CP) proposed a method that was so efficient that one could for the first time perform first-principles molecular dynamics simulations. Their key idea was to use molecular dynamics, not only to sample the atomic positions but also to minimize in practice the Kohn-Sham functional. Furthermore they used an efficient manipulation of the wave-functions in a plane-wave basis-set to speed up their calculations. Although nothing in the CP method is specific to a given type of basis-set, the truth is that the overwhelming number of CP simulations use a plane-wave basis-set, to the point that most people would automatically assume that a CP simulation would use a plane wave basis-set.

Although one can use plane-waves to calculate molecular properties with a super-cell method, most quantum chemists prefer the use of gaussian basis-sets. What we present here is an efficient parallel implementation of a method where the statistical sampling of the atomic positions is done with MC and the Kohn-Sham functional is directly minimized in a gaussian basis-set.

3 Numerical implementation

3.1 Construction of the matrix

Each matrix-element H_{ij} has many terms, which are usually classified by the number of different centers involved in its computation. The time and memory consuming terms are those associated with three center integrals used for the calculation of the effective potential v_{eff}. For the sake of simplicity we will assume that the effective potential is described also as a linear combination of functions $g_k(\boldsymbol{r})$,

$$v_{\text{eff}}(r, \{\psi_i\}) = \sum_{k=1}^{L} f_k(\{c_{ij}\})\, g_k(\boldsymbol{r}), \tag{9}$$

where the coeficients f_k have a dependence on the wavefunction coefficients, and g_k are atom centered gaussian functions. Actually, in the program only the exchange and correlation term of the effective potential is expanded this way, but the strategy of parallelization for all other contributions is exactly the same, and so we will not describe in detail the other terms.

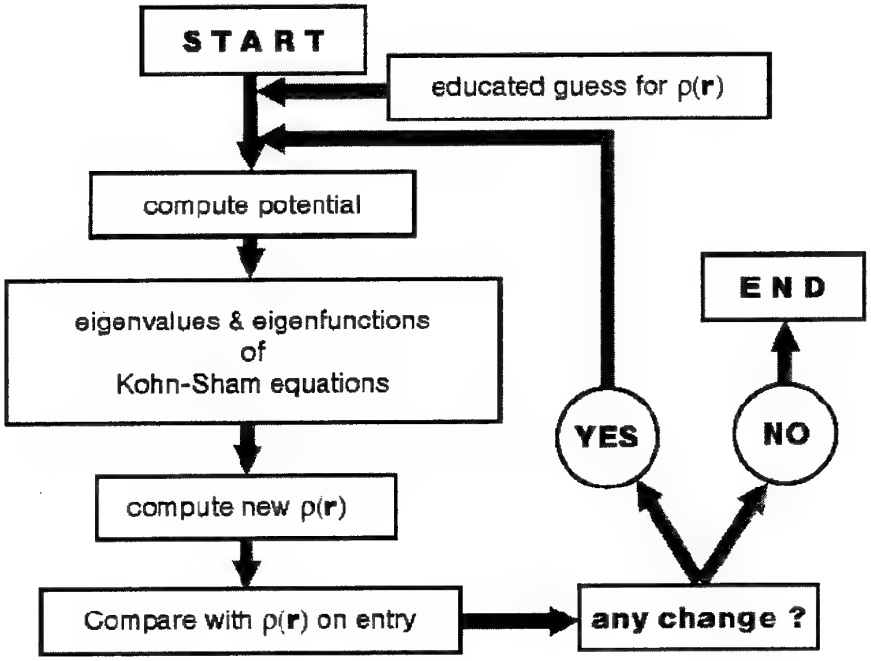


Fig. 1. self-consistent iterative scheme for solving the Kohn-Sham equations. One starts from an educated guess for the initial density which, in DFT, can be written in terms of the eigenfunctions of the Kohn-Sham equations as $\rho(\boldsymbol{r}) = \sum_i |\psi_i(\boldsymbol{r})|^2$. After several iterations one arrives at a density which does not change any more upon iteration.

The contribution of the effective potential to the hamiltonian H_{ij} is

$$V_{ij} = \int \phi_i(\boldsymbol{r}) v_{\text{eff}}(\boldsymbol{r}, \{\psi_i\}) \phi_j(\boldsymbol{r}) d^3r = \sum_{k=1}^{L} f_k(\{c_{ij}\}) \int \phi_i(\boldsymbol{r}) g_k(\boldsymbol{r}) \phi_j(\boldsymbol{r}) d^3r$$
$$= \sum_{k=1}^{L} f_k(\{c_{ij}\}) A_{ikj} \tag{10}$$

where the integral $A_{ikj} = \int \phi_i(\boldsymbol{r}) g_k(\boldsymbol{r}) \phi_j(\boldsymbol{r}) d^3r$ involves three gaussian functions, and can be calculated analytically. Furthermore all dependence on wavefunction coefficients is now in the coefficients f_k of the potential, and the integrals A_{ikj} are all the same in the self-consistent iterations. This means that all the iterative procedure illustrated in fig. 1 amounts now to recombine repeatedly the same integrals, but with different coefficients at different iterations throughout the self-consistent procedure.

We can now appreciate the two computational bottlenecks of a gaussian program. As the indexes i, j and k can reach to several hundred the size of the three-index array A_{ikj} requires a huge amount of memory. Although analytical, the calculation of each of the A_{ikj} is non-trivial and requires a reasonable number of floating point operations. The summation in eq. 10 has to be repeated for each of the self-consistent iterations.

So far, no parallelization has been attempted. We now use at profit the modular structure of the program in order to distribute tasks among the available processes in an even and non-overlapping way. In keeping with this discussion, we recast each matrix-element V_{ij} in the form

$$V_{ij} = \sum_{\lambda=1}^{N_{\text{proc}}} V_{ij}[\lambda] \tag{11}$$

where the indexed $V_{ij}[\lambda]$ will be evenly distributed among the N_{proc} processes executing the program, that is, it will be null except in one of the processes. Similarly, the three-index array A_{ikj} is distributed as

$$A_{ikj} = \sum_{\lambda=1}^{N_{\text{proc}}} A_{ikj}[\lambda] \tag{12}$$

in such a way that $A_{ikj}[\lambda]$ is null if $V_{ij}[\lambda]$ is null. Of course, the null elements are not stored so the large array is distributed among all the processes, which for a distributed memory machine means that A_{ikj} is distributed among all the processes. As

$$V_{ij}[\lambda] = \sum_{k=1}^{L} f_k(\{c_{ij}\}) A_{ikj}[\lambda] \tag{13}$$

there is no need to exchange the values of A_{ikj} among processes, but only those of f_k before summation, and $V_{ij}[\lambda]$ after the summation. So the calculation of

A_{ikj} is distributed among the processes, the storage is also distributed, and A_{ikj} never appears in the communications.

Finally, and due to the iterative nature of the self-consistent method, the code decides - *a priori* - which process will be responsible for the computation of a given contribution to $V_{ij}[\lambda]$. This allocation is kept unchanged throughout an entire self-consistent procedure.

3.2 Eigenvalue problem

For N_{at} atoms and, assuming that we take a basis-set of M gaussian functions per atom, our eigenvalue problem, eqs. 6 and 7, will involve a matrix of dimension $(N_{at} \times M)$. Typical numbers for an atomic cluster made out of 20 sodium atoms would be $N_{at} = 20$ and $M = 7$. This is a pretty small dimension for a matrix to be diagonalized, so the CPU effort is not associated with the eigenvalue problem but, *mostly*, with the construction of the matrix-elements H_{ij}. We have not yet parallelized this part of the code. Its paralellization, poses no conceptual difficulty, since this problem is taylor made to be dealt with by existing parallel packages, such as SCALAPACK. As this part of the code is the most CPU time consuming among the non-paralelized parts of the code, it is our next target for parallelization.

3.3 Monte-Carlo iterations

Once $E_{GS}(\boldsymbol{R}_1, \ldots, \boldsymbol{R}_{N_{at}})$ is obtained for a given molecular configuration, the Monte-Carlo Simulated Annealing algorithm "decides" upon the next move. As stated before, this procedure will be repeated many thousands of times before an annealed struture is obtained, hopefully corresponding to the global minimum of E_{GS}.

When moving from one MC iteration to the next, the Simulated Annealing algorithms typically change the coordinates of one single atom $\boldsymbol{R}_\alpha \rightarrow \boldsymbol{R}_\alpha + \delta\boldsymbol{R}$. As the basis set is localized, each of the indices in A_{ijk} is associated with a given atom. If none of the indices is associated with the atom $\boldsymbol{R}_\alpha$, than A_{ijk} does not change, and therefore is not recalculated. In this way, only a fraction of the order of $1/N_{\mathrm{at}}$ of the total number of integrals A_{ijk} needs to be recalculated, leading to a substantial saving in computer time, in particular for the larger systems ! Furthermore, the "educated guess" illustrated in fig. 1, used to start the self-consistent cycle is taken, for MC iteration $n + 1$, as the self-consistent density obtained from iteration n. In this way, in *all* but the start-up MC iteration, the number of iterations required to attain self-consistency becomes small. It is this coupling between the Monte-Carlo and DFT parts of the code that allow us to have a highly efficient code which enables us to run simulations in which the self-consistent energy of a large cluster needs to be computed many thounsands of times (see below).

4 Results and discussion

The program has been written in FORTRAN 77 and we use MPI as the underlying communication layer, although a PVM translation would pose no conceptual problems. Details of the DFT part of the program in its non-parallel version have been described previously ref[6]. The MC method and the SAM algorithm are well-described in many excellent textbooks[7].

The Hardware architecture in which *all* results presented here have been obtained is assembled as a farm of 22 DEC 500/500 workstations. The nodes are connected via a fast-ethernet switch, in such a way that all nodes reside in the same virtual (and private) fast-ethernet network. In what concerns Software, the 22 workstations are running Digital Unix version **4.0-d**, the DEC Fortran compiler together with DXML-libraries, and the communication layer is provided by the free MPICH[8] distribution, version **1.1**. Nevertheless, we would like to point out that the same program has been tested successfully on a PC, a dual-Pentium II-300, running Linux-SMP, g77-Fortran and LAM-MPI[9] version **6.2b**.

We started to test the code by choosing a non-trivial molecule for which results exist, obtained with other programs and using algorithms different from the SAM. Therefore, we considered an atomic cluster made out of eight sodium atoms - Na_8. Previous DFT calculations indicate that a D_{2d} structure - left panel of fig. 3 - corresponds to the global minimum of E_{GS}[6].
Making use of our program, we have reproduced this result without difficulties. Indeed, we performed several SAM runs starting from different choices for the initial structure, and the minimum value obtained for E_{GS} corresponded, indeed, to the D_{2d} structure. One should note that one SAM run for Na_8 involves the determination of E_{GS} up to $2,2\ 10^4$ times. Typically, we have used 1000 MC-iterations at a given fixed-temperature T in a single SAM run. This number, which is reasonable for the smaller clusters, becomes too small for the larger, whenever one wants to carefully sample the phase-space associated with the $\{\boldsymbol{R}_1, \ldots, \boldsymbol{R}_{N_{at}}\}$ coordinates.

As shown in the right panel of fig. 2, Na_9^+ was our second choice. This is a nine atom sodium cluster to which one electron has been removed. As is well known[5] this cluster, together with Na_8, constitute so-called magic clusters, in the sense that they display an abnormally large stability as compared to their neighbours in size[10]. When compared with quantum-chemistry results, the DFT structures are different, both for Na_8 and Na_9^+. This is not surprising, since the underlying theoretical methods and the minimization strategies utilized are also different, at the same time that the hyper-surface corresponding to $E_{GS}(\{\boldsymbol{R}_i\})$ is very shallow in the neighbourhood of the minima, irrespective of the method. Nevertheless, recent experimental evidence seem to support the DFT results[10].

In order to test the performance of the parallelization, we chose Na_9^+ and carried out two different kinds of benchmarks. First we executed the program performing 1 iteration - the start-up iteration - for Na_9^+ and measured the CPU time T_{CPU} as a function of the number of processes N_{PROC}. For the basis-set used, the number of computed A_{ikj} elements is, in this case 3321. As can be seen from eq. 13, the ratio of computation to communications is proportional to

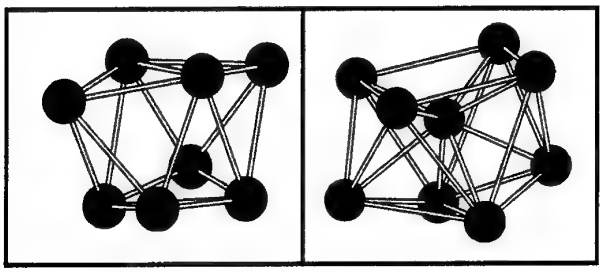


Fig. 2. global minimum of E_{GS} for the two magic sodium clusters Na_8 and Na_9^+. For the determination of such global minima a *SAM* algorithm has been employed, requiring many thousands of first-principles computations of E_{GS} to be carried out.

the number of fit functions L. By choosing a small molecule where L is small we are showing an unfavorable case, where the parallelization gains are small, so we can discuss the limits of our method. In fig. 3 we plot, with a solid line, the inverse of the CPU time as a function of N_{PROC}.

Our second benchmark calculation involves the computation of 100 MC-iterations. For direct comparison within the same scale, we multiplied the inverse of T_{CPU} by the number of iterations. The resulting curve is drawn with a dashed line in fig. 3.

Several features can be inferred from a direct comparison of the 2 curves. First of all, there is an ideal number N_{PROC} into which the run should be distributed. Indeed, fig. 3 shows that efficiency may actually drop as N_{PROC} is increased. For this particular system, $N_{\mathrm{PROC}} = 8$ is the ideal number. This "node-saturation" which takes place here for Na_9^+ is related to the fact that the time per iteration is small enough for one to be able to observe the overhead in communications due to the large number of nodes in which the run is distributed. When the number of atoms increases, this overhead becomes comparatively smaller and ceases to produce such a visible impact on the overall benchmarks. From fig. 3 one can also observe that, for small N_{PROC} , the largest gain of efficiency is obtained for the 1-iteration curve. This is so because that is where the parallelization plays a big role. Indeed, as stated in section 3, the number of floating point operations which are actually performed in the subsequent MC-iterations is considerably reduced, compared to those carried out during the start-up iteration. As a result, the relative gain of efficiency as N_{PROC} increases becomes smaller in this case. However, since both CPU *and* memory are distributed, it may prove convenient to distribute a given run, even if the gain is not overwhelming.

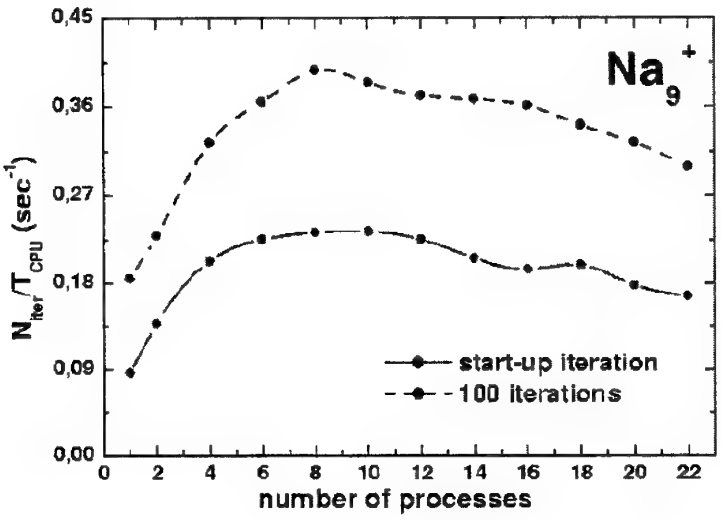


Fig. 3. Dependence of inverse CPU time (multiplied by the number of *MC*-iterations) as a function of the number of processes (in our case, also dedicated processors) for two benchmark calculations (see main text for details). A direct comparison of the curves illustrates what has been parallelized in the code and where the parallelization plays its major role.

The solid curve of fig. 3 is well fitted by the function $0,25 - 0,17/N_{\text{proc}}$ up to $N_{\text{proc}} = 8$ which reveals that a good level of parallelization has been obtained. This is particularly true if we consider that the sequential code has 14200 lines, and is very complex, combining many different numerical algorithms.

Finally, we would like to remark that, at present, memory requirements seem to put the strongest restrictions on the use of the code. This is so because of the peculiar behaviour of MPICH which creates, for each process, a "clone-listener" of each original process, that requires the *same* amount of memory as the original processes. This is unfortunate since it imposes, for big molecules, to set up a very large amount of swap space on the disk in order to enable MPI to operate successfully. In our opinion, this is a clear limitation. We are, at present, working on alternative ways to overcome such problems.

In fig. 4 we show our most recent results in the search for global minima of sodium clusters. The structures displayed in fig. 4 have now 21 (left panel) and 41 (right panel) sodium atoms. A total of 17955 matrix-elements is required to compute each iteration of the self-consistent procedure for Na_{21}^{+} whereas for Na_{41}^{+} the corresponding number is 68265. The structures shown in fig. 4 illustrate the possibilities of the code, which are, at present limited by swap limitations exclusively. Of course, the CPU time for these simulations is much bigger than for the smaller clusters discussed previously. In this sense, the structure shown for Na_{41}^{+} cannot be considered unambiguosly converged, in the sense that more SAM runs need to be executed. On the other hand, we believe the structure depicted for Na_{21}^{+} to be fully converged. Since no direct experimental data for these structures exists, only indirect evidence can support or rule out such structural optimizations. The available experimental data[10] indirectly supports this structure since, from the experimental location of the main peaks of the photo-absorption spectrum of such a cluster one may infer the principal-axes ratio of the cluster, in agreement with the prediction of fig. 4.

5 Conclusions and future applications

In summary, we have suceeded in parallelizing a DFT code which efficiently computes the total energy of a large molecule. We have managed to parallelize the most time and memory consuming parts of the program, except, as mentioned in section 3.2, the diagonalization block, which remains to be done. This is good enough for a small farm of workstations, but not for a massive parallel computer. We should point out that it is almost trivial to parallelize the Monte-Carlo algorithm. In fact as a SAM is repeated starting from different initial configurations, one just has to run several jobs simultaneously, each in its group of processors. However, this will not have the advantages of distributing the large matrix A_{ijk}. As storage is critical for larger molecules, parallelizing the DFT part of the code may be advantageous even when the gains in CPU time do not look promising.

The code is best suited for use in combination with MC-type of simulations, since we have shown that, under such circumstances, not only the results of a

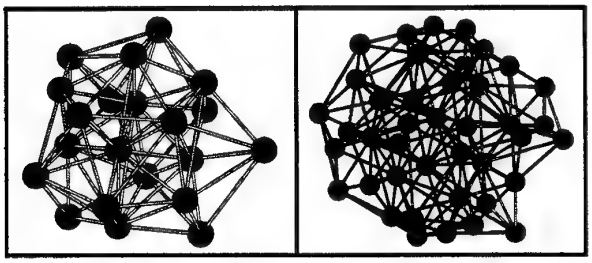


Fig. 4. Global minima for two large singly ionized sodium clusters with 21 atoms (left panel) and 41 atoms (right panel). Whereas the structure of Na_{21}^{+} can be considered as "converged", the same cannot be unambiguously stated for the structure shown for Na_{41}^{+}. For this largest cluster, the structure displayed shows our best result so-far, although further *SAM* runs need to be carried out.

given iteration provide an excellent starting point for the following iteration, but also the amount of computation necessary to compute the total energy at a given iteration has been worked out, to a large extent, in the previuous iteration. Preliminary results illustrate the feasibility of running first-principles, large-scale *SAM* simulations of big molecules, without resorting to dedicated supercomputers. Work along these lines is under way.

Acknowledgements

JMP and JLM acknowledge financial support from the Ministry of Science and Technology under contracts PRAXIS / C / FIS / 10019 / 1998 and PRAXIS / 2 / 2.1 / FIS / 26 / 94, respectively.

References

1. R.M. Dreizler, E.K.U. Gross *Density Functional Theory* (Springer-Verlag, Berlin 1990) ;
2. G. D. Mahan, K. R. Subbaswamy, *Local Density Theory of the Polarizability* (Plenum Press, 1990) ;
3. F. Alasia et al., *J. Phys.* **B27** (1994) L643 ;
4. R. Car, M. Parrinello, *Phys. Rev. Lett.* **55** (1985) 2471 ;
5. J. M. Pacheco, Walter P. Ekardt, *Annalen der Physik (Leipzig)*, **1** (1992) 254 ;
6. J. L. Martins, R. Car, J. Buttet, *J. Chem. Phys.* **78** (1983) 5646; J. L. Martins, J. Buttet, R. Car *Phys. Rev.* **B31** (1985) 1804;

7. W. M. Press, B. P. Flannery, S. A. Teukolsky, W. T. Vetterling, *Numerical Recipes* 2nd edition, (Cambridge University Press, 1990) ; H. Gould, J. Tobochnik, *An introduction to computer simulation methods*, 2nd edition, (Addison-Wesley, 1996) ;
8. W. Gropp, E. Lusk, N. Doss, A. Skjellum, *Parallel Computing*, **2** (1996) 789 ; W. Gropp, E. Lusk, *User's Guide for* `mpich`, *a Portable Implementation of MPI*, (Mathematics and Computer Science Division, Argonne National Laboratory, 1996) (http://www-unix.mcs.anl.gov/mpi/mpich/)
9. LAM Project, Laboratory for Scientific Computing, University of Notre Dame, U.S.A. (lam@mpi.nd.edu, http://www.mpi.nd.edu/lam)
10. Walter P. Ekardt (Editor), *Metal Clusters*, Wiley Series in Theoretical Chemistry, (John Wiley & Sons, 1990) ;

An Efficient Parallel Algorithm for the Numerical Solution of Schrödinger Equation.

Jesús Vigo-Aguiar[1], Luis M. Quintales[2] and S. Natesan[3]

[1] Dept. of Mathematical Sciences, University of Wisconsin-Milwaukee.
PO Box 413, Milwaukee, WI, 53201, USA.
jvigo@gugu.usal.es.
[2] Dept. of Informatica, University of Salamanca.
E-37008 Salamanca, Spain.
lamq@gugu.usal.es.
[3] Dept. of Mathematics, Bharathidasan University,
Tiruchirappalli 620 024, Tamilnadu, INDIA.
matnat@bdu.ernet.in

Abstract. In this paper we show how to construct parallel explicit multistep algorithms for an accurate and efficient numerical integration of the radial Schrödinger equation. The proposed methods are adapted to Bessel functions, that is to say, they integrate exactly any linear combination of Bessel and Newman functions and ordinary polynomials. They are the first of the like methods that can achieve any order. The coefficients of the method are computed in each step. We show how the parallel implementation of the method is the key of an efficient computation.

Corresponding author: J. Vigo-Aguiar

1 Introduction.

The behavior of a spinless quantum particle of mass m in a potential $v(X)$, $X = (x_1, x_2, x_3)$ is governed by the three-dimensional Schrödinger equation

$$\frac{-h^2}{2m}\Delta y(X) + (v(X) - \epsilon)y(X) = 0 \tag{1}$$

where Δ is the Laplace operator, h is the reduced Planck's constant and ϵ is the particle energy. The solution $y(X)$ can be expanded on the complete set of spherical functions $Y_{l,m}$

$$y(X) = \frac{1}{x}\sum_{l=0}^{\infty}\sum_{m=-l}^{l} y_l(x)Y_{l,m}(\theta,\rho) \tag{2}$$

where x, ρ, θ are the spherical coordinates of the point $\boldsymbol{x}$. Introducing this expansion in the equation and operating, we find that $y_l(x)$ satisfies

$$y_l''(x) = (U(x) - P(x))\, y_l(x), \tag{3}$$

where

$$k^2 = \frac{2m}{h^2}\epsilon \quad P(x) = k^2 - \frac{l(l+1)}{x^2} \tag{4}$$

and $U(x)$ is a given potential. The solution of the equation must vanish at the origin, i.e. one boundary condition is $y_l(0) = 0$, and the other boundary condition, which depends on the physical model, is imposed at large x.

Equation (3) is usually know as radial Schrödinger equation. And the problem of integrating (1) has been transformed to the integration of a infinite set of second order differential equations. Then it is obvious that we need methods with small CPU times. The use of parallel procedures and adequate multistep methods allow fast and accurate integration.

In the computation of the eigenvalues or the phase shifts of the radial Schrödinger equation, usually the potential $U(x)$ tends to zero much faster than the centrifugal potential $k^2 - P(x) = l(l+1)/x^2$ and then the solution of (3) may be taken as

$$y(x) \equiv cte_1 kx j_l(kx) + cte_2 kx n_l(kx) \tag{5}$$

where $j_l(x)$ and $n_l(x)$ are respectively the Spherical Bessel and Neumann functions. It is our intention to develop a method that integrates exactly any linear combination of this functions and ordinary polynomials. This property is known as Bessel fitting or adaptation to Bessel functions. The theory and a procedure to construct adapted multistep methods to trigonometric and exponential functions is nowadays solved and can be found in [8]. Theory and procedure for adaptation to other types of dynamic behavior is still an open question.

The difficulty of construction of methods adapted to Bessel functions is evidenced by the fact that there exist only a few satisfactory papers on the subject (see for example, Raptis and Cash [3] and Simos and Raptis [5]) The methods of Raptis and Cash produce accurate solutions in the phase shift problem that they proposed in spite of being second and fourth order methods. However in their methods the coefficients depend on the point where we are calculating the solution and so they must be recalculated at every step, with high computational cost. This is the point where parallel implementation is fundamental. It is our goal to formulate higher order Bessel fitting methods with the possibility that the coefficients can be computed at the beginning of the program in parallel, thus allowing a significant reduction in the computational cost.

2 Bessel and Neumann fitting methods.

To construct our procedure let us consider the differential equation

$$y'' + P(x)y = f(x, y) \tag{6}$$

Our first observation is that the sequence $y_n = cte_1 k\ n\ hj_l(nh) + cte_2 k\ n\ hn_l(nh)$ is the solution of the difference equation

$$\begin{aligned} &y_{n+1} + d_l(x_{n+1}, x_{n-1}, h)y_n + d_l(x_{n+1}, x_n, h)y_{n-1} = 0 \\ &y_0 = 0 \\ &y_1 = cte_1 k\ hj_l(h) + cte_2 k\ hn_l(h) \end{aligned} \tag{7}$$

where

$$d_l(a,b,h) = \frac{\begin{Vmatrix} kaj_l(ka) & kbj_l(kb) \\ kan_l(ka) & kbn_l(kb) \end{Vmatrix}}{\begin{Vmatrix} k(a-h)j_l(a-h) & k(a-2h)j_l(a-2h) \\ k(a-h)n_l(a-h) & k(a-2h)n_l(a-2h) \end{Vmatrix}} \tag{8}$$

and $\| \ \|$ denotes the determinant.

Then the problem

$$y'' + P(x)y = 0 \tag{9}$$

is integrated exactly with the proposed difference equation.

The construction of the discretization scheme is completed with the treatment of the right-hand side $f(x,y)$ in (6). In the theory of classical multistep methods, $f(x,y)$ is approximated by a interpolatory polynomial in the previous steps. The same proceeding is done here. The expression of the Bessel fitted method applied to (6) is:

$$y_{n+1} + d_l(x_{n+1},x_{n-1},h)y_n + d_l(x_{n+1},x_n,h)y_{n-1} = h^2 \sum_{i=0}^{k} \alpha_i f(x_{n+1-i}, y_{n+1-i}) \tag{10}$$

We impose that the method integrates exactly the interpolation polynomial of $f(x,y)$ requiring the method to be exact when we integrate the equations

$$y''(x) + P(x)y = P(x)x^m + m(m-1)x^{m-2}, \tag{11}$$

for $m = 0, 1, \cdots, k$. With this condition we obtain the nonsingular system of linear equations for α_i:

$$A\alpha = Q \tag{12}$$

where A is

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ x_{n+1} & x_n & \cdots & x_{n-k} \\ 2q(x_{n+1}) + x_{n+1}^2 & 2q(x_n) + x_n & \cdots & 2q(x_{n-k}) + x_{n-k} \\ \cdots & \cdots & \cdots & \cdots \\ k(k-1)x_{n+1}^{k-2}q(x_{n+1}) + x_{n+1}^k & k(k-1)x_n^{k-2}q(x_n) + x_n^k & \cdots & k(k-1)q(x_{n-k})x_{n-k}^{k-2} + x_{n-k}^k \end{pmatrix} \tag{13}$$

and

$$\alpha = \begin{pmatrix} h^2P(x_{n+1})\alpha_0 \\ h^2P(x_{n+1})\alpha_1 \\ \vdots \\ h^2P(x_{n+1})\alpha_k \end{pmatrix} \quad Q = \begin{pmatrix} q_0 \\ q_1 \\ \vdots \\ q_k \end{pmatrix} \tag{14}$$

where $q(x) = \dfrac{1}{P(x)}$,

$$q_i = x_{n+1}^i + d_l(x_{n+1},x_{n-1},h)x_n^i + d_l(x_{n+1},x_n,h)x_{n-1}^i. \tag{15}$$

The solution of this system of equations can be done for $0 < k < 10$ with the help of a symbolic manipulator. The resulting scheme will be named PSBF (Parallel Spherical Bessel Fitted method) in the following. Note that the coefficients α are recalculated once in each step. That is a characteristic of all the methods that integrate exactly linear differential equations without constant coefficients.

Our method is an implicit method, in the same way we could have deduced an explicit method. However when we apply our method to equation (3) we can obtain an explicit procedure:

$$y_{n+1} = \frac{-1}{1-\alpha_0\, U(x_{n+1})}(d_l(x_{n+1}, x_{n-1}, h)y_n + d_l(x_{n+1}, x_n, h)y_{n-1} \\ -h^2 \sum_{i=1}^{k} \alpha_i V(x_{n+1-i})y_{n+1-i} \tag{16}$$

Given the good properties of stability of the implicit methods we have considered unnecessary to use an explicit method.

3 Parallel Implementation and Properties.

We will give a brief explanation of the convergence of the method (detailed proofs will appear in a different paper).

Theorem The multistep method PSBF of $k+1$ steps is consistent of order $k+1$. Its local truncation error can be expressed as

$$\mathcal{L}_P(y, h)(x) = h^{k+3}\alpha_{k+1}P(D)y(x) + O(h^{k+4}) \tag{17}$$

where $P(D)y(x)$ is certain combination of $y(x)$ and its derivatives. The method integrates without local truncation error the problems (3) whose solution belongs to the space generated by the linear combinations of

$$1,\ x,\ x^2,\ \cdots, x^{k_p},\ xj_l(x), xn_l(x) \tag{18}$$

Observe that this method reduces to the classical Cowell method (Henrici 1962) when $P(x) = 0$. For $k = 2$ the methods reduces to the popular Numerov method.

What makes the method different from standard methods is that the coefficients are recalculated in each step. This fact produces an increase in the computational cost, however this cost is minimized if we use the parallel implementation proposed in this paper.

We observe that once the grid has been selected the coefficients α_j, d_1, and d_2, can be calculated independently at each point x_n. Then at the beginning of the program we compute in parallel these coefficients for all the points x_n of the grid. In the same way we compute all the values of the potential at each point at the beginning of the program. We have called this phase initialization phase. The following diagram explains the idea (see Table 1).

Table 1. Diagram for the initialization. Order of the method k, number of total steps in the integration m n

Processor 1	$\cdots$	Processor m
d_1, d_2	$\cdots$	d_1, d_2
$\alpha_1 \cdots \alpha_k$	$\cdots$	$\alpha_1 \cdots \alpha_k$
at the points $x_1 \cdots x_n$	$\cdots$	at the points $x_{(n-1)m+1} \cdots x_{nm}$

Table 2 shows the execution time of this initial processes for a method of order $k = 6$ and total number of steps 296. We integrate a single equation and a system of dimension 80 ($l = 0 \cdots 79$). We show how the speed-up is close to the the number of processors. When we are using a scheme with constant coefficients the CPU time of the initialization is only due to the computation of the potential in the grid.

Table 2.

num. of processors	T/CPU (1 Eq.)	Speed-up	T/CPU (80 Eq.)
2	0.0254 sec.	1.92	2.0 sec.
3	0.0183 sec.	2.54	1.5 sec.
4	0.0143 sec.	3.08	1.1 sec.

The following figures show a snapshot of the initialization of the parallel process. The green color represents computation time of each processor. The yellow/red color represents communication times. The green zone in the processor 0 represents the integration. It can be observed that the final speedup is roughly related to the ratio between the green and yellow areas during the initialization phase.

We would like to point out in this section that if the equation we are integrating needs an explicit method for its computation, a predictor and a corrector method can be obtained using the following recurrences

$$\begin{array}{l} y^p_{n+4} + d_l(x_{n+4}, x_n, 2h) y^c_{n+2} + d_l(x_{n+4}, x_{n+2}, 2h) y^c_n \\ y^c_{n+3} + d_l(x_{n+3}, x_{n+1}, h) y^c_{n+2} + d_l(x_{n+3}, x_{n+2}, h) y^c_{n+1} \end{array} \tag{19}$$

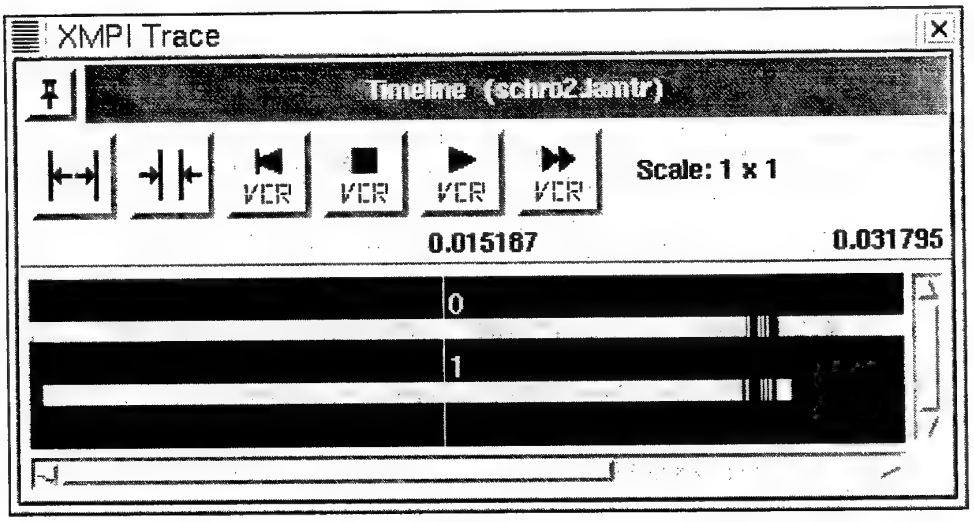


Fig. 1. Parallel execution snapshot with 2 processors

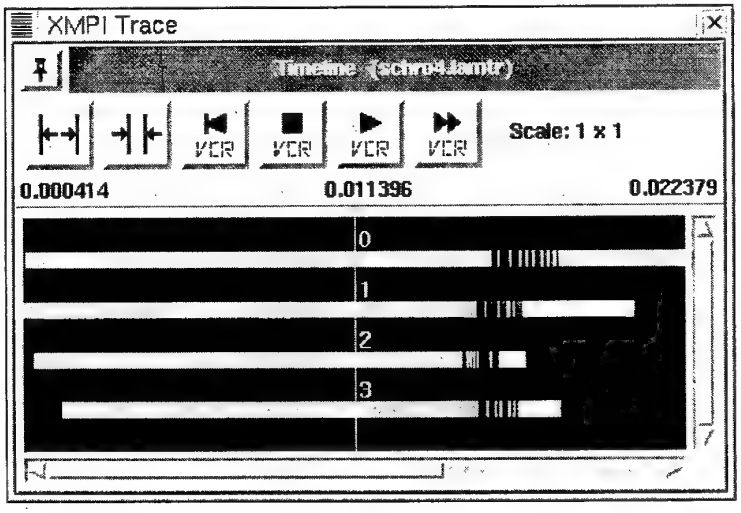


Fig. 2. Parallel execution snapshot with 4 processors

These recurrences and the procedure mentioned in section 2 allows us to write the methods in the form

$$\begin{aligned} & y^p_{n+4} + d_l(x_{n+4}, x_n, 2h) y^c_{n+2} + d_l(x_{n+4}, x_{n+2}, 2h) y^c_n = \\ & = h^2 \alpha_0 f^p_{n+3} + \sum_{i=1}^{k} \alpha_i f(x_{n+1-i}, y^p_{n+1-i}) \\ & y^c_{n+3} + d_l(x_{n+3}, x_{n+1}, h) y^c_{n+2} + d_l(x_{n+3}, x_{n+2}, h) y^c_{n+1} = \\ & = h^2 \beta_0 f^p_{n+3} + \sum_{i=1}^{k} \beta_i f(x_{n+1-i}, y^p_{n+1-i}), \end{aligned} \tag{20}$$

where the coefficients α_i and β_i are solution of a system of equation similar to (12). The implementation in this case is similar to the one proposed in [9].

4 Numerical Examples

In order to test the accuracy of the proposed procedure we apply it to the solution of equation (3) using as $U(x)$ the Leonard-Jones potential which has been widely discussed in literature. For this problem the potential has been taken as in Simos and Raptis

$$U(x) = m(\frac{1}{x^{12}} - \frac{1}{x^6}) \tag{21}$$

where $m = 500$.

The considered problem is the computation of the relevant phase shifts. We initialize the integration with the popular Numerov method using a small step. We do not take in account the step given by the Numerov method in the results presented.

We consider (following for example T. Simos) the asymptotic form of the solution

$$\begin{aligned} & y(x) \approx Akxj_l(kx) - Bkn_l(kx) \approx AC(\sin(kx - \tfrac{\Pi}{2}) \\ & + \tan\delta_l \cos(kx - \tfrac{\Pi}{2})) \end{aligned} \tag{22}$$

where δ_l is the phase shift that may be calculated from the formula

$$\tan\delta_l = \frac{y(x_2)S(x_1) - y(x_1)S(x_2)}{y(x_1)C(x_2) - y(x_2)C(x_1)} \tag{23}$$

for x_1 and x_2 distinct points on the asymptotic region. We take as asymptotic region $x \geq 15$ and $x_1 = 15$ and $x_2 = 15 - h$, h being the step size. Here $S(x) = kxj_l(kx)$ and $C(x) = kxn_l(kx)$.

Since the problem is treated as an initial-value problem, one needs y_0 and y_1 before starting the Numerov method. As we have mentioned $y_0 = 0$, and following [4, 1] the solution behaves as constant by x^{l+1} as $x \to 0$. According to this we take $y_1 = h^{l+1}$.

In the next table, we have chosen $k = 5$ and we represent the error with respect to the true phase shift of the proposed method using order 6, the results can be compared with those obtained by Simos [5].

Table 3. k=5. Accuracy in phase shift. Order 6. Number of steps 292, $h = 0.05$

l	True Phase shift	Computed Phase shift	Error
0	-0.4831	-0.4832	10^{-4}
1	0.9282	0.9277	$5\ 10^{-4}$
2	-0.9637	-0.9639	$2\ 10^{-4}$
3	0.1206	0.1170	$36\ 10^{-4}$
4	1.0328	1.0349	$21\ 10^{-4}$
5	-1.3785	-1.3779	$6\ 10^{-4}$
6	-0.8441	-0.843	$8\ 10^{-4}$
7	-0.5244	-0.5256	$12\ 10^{-4}$
8	-0.4575	-0.4575	–
9	-0.7571	-0.7571	–
10	1.4148	1.4148	–

All computations were carried out on a Silicon Graphics Origin 200 Server with four processors MIPS R10000 and the MPI library LAM 6.3 [2]. In the present architecture communication is an operation of write/read using the shared memory. We have used FORTRAN and Double precision arithmetic with 16 digits accuracy.

Conclusion: As we can see, the fact that we need to compute all the coefficients in each step means a computational cost of few seconds, even if we are working with big systems of ODEs. However we are obtaining a significant improvement in the precision. In the opinion of the authors, the effectiveness of the method proposed in this work has been demonstrated since parallel machines with at least a few processors are nowadays quite commonly available.

5 Acknowledgments:

This work was supported by Junta of Castilla y León under the Project SA 69/99.

References

1. J.R. Cash and A.D. Raptis Computer Physics Communications, **33** (1984), 299-304.
2. Dongarra J.J., Otto S. W., Snir M., Communications of the ACM. 39, (3) 1996, 84-90

3. A. D. Raptis and J. Cash. Exponential and Bessel Fitting Methods for the Numerical Solution of the Schrödinger Equation. Computer Physic Communications, **44** (1987) 95-103.
4. T. E. Simos. IMA J. Numer. Anal. **11** (1991), 347.
5. T. E. Simos and A. D. Raptis. A Four Order Bessel Fitting Method for the Numerical Solution of the Schrödinger Equation. *J. Comput. Appl. Math.* **43** (1992), 313-322.
6. E.L. Stiefel, and G. Scheifele. *Linear and Regular Celestial Mechanics.* Springer, Berlin-Heidelberg-New York (1971).
7. E. Stiefel and D.G. Bettis. *Numer. Math.* **13** (1969), 154.
8. J. Vigo-Aguiar and J. M. Ferrándiz. A general Procedure for the Adaptation of Multistep Algorithm to the integration of Oscillatory Problems. SIAM J. of Num. Anal., **12**.
9. J. Vigo-Aguiar and L.M. Quintales. A parallel ODE Solver Adapted to Oscillatory Problems. Proceedings of the 1999 international Conference on Parallel and Distributed Processing Technology and Applications. H. Arabnia Editor, CSREA Press, Athens GA 1999. 233–239.

An Efficient Parallel Algorithm for the Symmetric Tridiagonal Eigenvalue Problem

Maria Antónia Forjaz[1] and Rui Ralha

Departamento de Matemática
Universidade do Minho
Campus de Gualtar
4710-057 Braga, Portugal
Tel +351 253604340, Fax +351 253678982
maf@math.uminho.pt, r_ralha@math.uminho.pt

Abstract. An efficient parallel algorithm, *farmzeroinNR*, for the eigenvalue problem of a symmetric tridiagonal matrix is implemented in a distributed memory multiprocessor with 112 nodes [For00]. The basis of our parallel implementation, is an improved version of the *zeroinNR* method [Ral93]. It is consistently faster than simple bisection and produces more accurate eigenvalues than the QR method. As it happens with bisection, *zeroinNR* exhibits great flexibility and allows the computation of a subset of the spectrum with some prescribed accuracy. Results were carried out with matrices of different types and sizes up to 10^4 and show that our algorithm is efficient and scalable.

1 Introduction

The computation of the eigenvalues of symmetric tridiagonal matrices is one of the most important problems in numerical linear algebra. The reason for this is the fact that in many cases the initial matrix, if not already in tridiagonal form, is reduced to this form using either orthogonal similarity transformations. in the case of dense matrices, or the Lanczos method, in the case of large sparse matrices.

Essentially we can consider three different kinds of methods for this problem: the QR method and their variations [Par80], [Dem97], the divide-and-conquer methods[2] [Cup81], [DS87], and the bisection-multisection methods [Wil65], [RR78], [Par80], [Ber84]. The bisection method is a robust method but is slower than the other methods for the computation of the complete set of eigenvalues. However, because of the excellent opportunities it offers for parallel processing. several parallel algorithms have been proposed which use bisection to isolate each eigenvalue and then some additional technique with better convergence rate

[1] Candidate to the Best Student Paper Award

[2] Available as LAPACK routine sstevd; a good choice if we desire all eigenvalues and eigenvectors of a tridiagonal matrix whose dimension is larger than about 25 [Dem97, pg. 217].

to compute the eigenvalue to the prescribed accuracy [LPS87], [IJ90], [Kal90], [BW20], [DHvdV93]. One of such methods, dubbed *zeroinNR*, has been proposed in [Ral93] and uses an original implementation of the Newton-Raphson's method for this purpose.

2 A Sequential Algorithm: *zeroinNR*

Let A be a real, symmetric tridiagonal matrix, with diagonal elements $a_1, \ldots, a_n$ and off-diagonal elements $b_1, \ldots, b_{n-1}$. The sequence of leading principal minors of A is given by

$$\begin{cases} p_0(\lambda) = 1 \\ p_1(\lambda) = a_1 - \lambda \\ p_i(\lambda) = (a_i - \lambda)p_{(i-1)}(\lambda) - b_i^2 p_{(i-2)}(\lambda), \quad i = 2, 3, \ldots, n. \end{cases} \tag{1}$$

It is well known that *the number of variations of sign in this sequence equals the number of eigenvalues of A which are strictly smaller than λ.*

To avoid overflow problems, the sequence (1) can be modified to the form

$$\begin{cases} q_0(\lambda) = 1 \\ q_1(\lambda) = a_1 - \lambda \\ q_i(\lambda) = p_i(\lambda)/p_{i-1}(\lambda), \; i = 2, 3, \ldots, n \end{cases} \tag{2}$$

and the terms of the new sequence can be obtain by the following expressions.

$$\begin{cases} q_0(\lambda) = 1 \\ q_1(\lambda) = a_1 - \lambda \\ q_i(\lambda) = (a_i - \lambda) - b_i^2/q_{i-1}(\lambda), \; i = 2, 3, \ldots, n \end{cases} \tag{3}$$

where the number of negative terms $q_i(\lambda)$, $i = 0, \ldots, n$, is equal to the number of eigenvalues strictly smaller than λ. This is the basis for the bisection method implemented in [BM+67], which is known to have excellent numerical properties in the sense that it produces very accurate eigenvalues. The drawback of bisection is its linear convergence rate[3] that makes the method slower than others, at least for the computation of the complete system. Different authors have proposed modifications of the simple bisection method in order to accelerate its convergence. One such proposal, dubbed the *zeroinNR* method, has been given in [Ral93] and essentially uses Newton-Raphson's method to find an eigenvalue after it has been isolated by bisection. The correction $p_n(x_k)/p'_n(x_k)$, in the iterative formula of the Newton-Raphson method,

$$x_{k+1} \leftarrow x_k - \frac{p(x_k)}{p'(x_k)}, \tag{4}$$

is obtained without explicitly calculating the values of the polinomial $p_n(x_k)$, and its derivative $p'_n(x_k)$, therefore avoiding overflow and underflow in such

[3] The bisection method converges linearly, with one bit of accuracy for each step.

computations. For this purpose the following algorithm has been derived. From (2) we have,

$$p_i = q_i\, p_{i-1}$$

and by differentiation

$$p_i' = q_i'\, p_{i-1} + q_i\, p_{i-1}'$$

and carrying out the division by p_i, we obtain the following expression

$$\frac{p_i'}{p_i} = \frac{q_i'}{q_i} + \frac{p_{i-1}'}{p_i} \tag{5}$$

which relates the arithmetic inverses of the Newton-Raphson correction for the polynomials p_{i-1} and p_i, and their quotient q_i.

From the recursive expression, (3), we have that,

$$q_i' = -1 + b_i^2\, \frac{q_{i-1}'}{q_{i-1}^2}, \quad i = 2, 3, \ldots, n$$

and carrying out the division by q_i,

$$\frac{q_i'}{q_i} = \frac{-1}{q_i}\left(-1 + \frac{b_i^2}{q_{i-1}}\,\frac{q_{i-1}'}{q_{i-1}}\right), \quad i = 2, 3. \ldots, n$$

Using the notation

$$\Delta q_i = q_i'/q_i, \quad \Delta p_i = p_i'/p_i$$

the complete computation of,

$$\Delta p_n = p_n'(x)/p_n(x)$$

is expressed in the following equations,

$$\left.\begin{array}{l} q_1 = a_1 - x \\ \Delta q_1 = \Delta p_1 = -1/q_1 \\ q_i = a_i - x - b_i^2/q_{i-1} \\ \Delta q_i = (-1 + b_i^2/q_{i-1} * \Delta q_{i-1})/qi \\ \Delta p_i = \Delta q_i + \Delta p_{i-1} \end{array}\right\} i = 2, \ldots, n \tag{6}$$

where $\Delta q_i = q_i'(x_k)/q_i(x_k)$ and $\Delta p_i = p_i'(x_k)/p_i(x_k)$.

It is important to observe that in the computation of Δp_n using the formulae (6), the values $q_i, i = 1, \ldots, n$, are obtained, and its signs can be used to derive a method that combines bisection and Newton-Raphson's iteration. We will refer to this method as the *zeroinNR* algorithm.

So, given an interval $[\alpha, \beta]$ which contains an eigenvalue, and given an approximation $x_k \in [\alpha, \beta]$, the *zeroinNR* method will produce, in each step, an approximation x_{k+1} to the eigenvalue.

The *zeroinNR* method although not as fast as the QR method (according to [Ral93], *zeroinNR* is about two to four times slower than QR for the computation of all eigenvalues, depending on the characteristics of the spectrum) is consistently faster than simple bisection (generally, twice as fast) and retains the excellent numerical properties of simple bisection. In the present work we have introduced some modifications in the original *zeroinNR* method which actually make it faster. Numerical tests were carried out in a transputer based machine using double precision arithmetic. The methods were implemented in Occam 2, the official transputer's language.

We were able to find out the errors in the computed eigenvalues since we have used matrices for which analytic expressions for the eigenvalues are known. We conclude that, for small matrices, the accuracy of *zeroinNR* is comparable to that of the QR method as implemented in the MatLab system [Mat99], but as the size of the matrices grows, the *zeroinNR* method provides more accurate eigenvalues than QR method.

This can be appreciated in Figure 1, where the absolute erros of a matrix of size 1000. are plotted. We have used the tridiagonal matrix with $a_i = 2$ and $b_i = 1$ which eigenvalues are given by

$$\lambda_i = 2 + 2\cos\left(\frac{i\pi}{n+1}\right), \quad i = 1, \ldots, n.$$

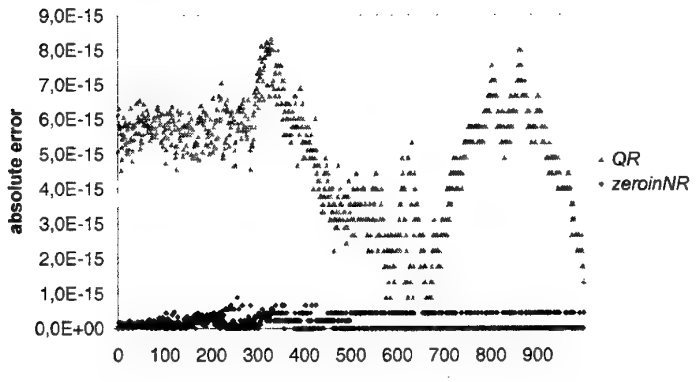


Fig. 1. Absolute errors of the eigenvalues of a matrix ($n = 1000$) computed with *zeroinNR* and QR.

3 An Efficient Parallel Algorithm: *farmzeroinNR*

The sequential *zeroinNR* method can be readily adapted to parallel processing since several disjoint intervals can be treated simultaneously by different processors. We have developed a parallel organization under a *processor farm* model

and we will refer to this parallel implementation as the *farmzeroinNR* method. The typical architecture for this model is a pipeline of processors (workers), where the master sends tasks to workers and gets back the results produced.

Each time a processor produces two disjoint intervals containing eigenvalues, as the result of a bisection step, it keeps only one of them and passes back to the *master* the second interval which is kept in a queue of tasks. As soon as there is an available *worker* somewhere in the line, a new task is fed into the pipeline. Because of this mechanism, the algorithm achieves dynamic load balancing.

A dynamic distribution of tasks results from the fact already mentioned, as soon as a *worker* finishes a task, it will get a new one from the queue (which is managed by *master*), if such queue is not empty. The advantage of such dynamic workload distribution gets more important as n grows. It must be noted that, because some tasks take longer to finish than others, workers may not execute the same number of tasks, but will spend about the same time working.

The pseudocode to the *master* and *worker* processors are given in Algorithm 1 and Algorithm 2, respectively.

```
eig ← 0
for k ← 1 .. p − 1 do
    { worker[k] ← initial_interval[k]
procs ← 0
while eig < n do
    { case input_channel is_a
        interval ⟶ { if procs > 0 do
                         { output ← interval to workers
                           procs ← procs − 1
                     else ⟶
                         { queue ← interval
        eigenvalue ⟶ { eig ← eig + 1
                       if queue not empty do
                           { output ← interval to workers
                       else ⟶
                           { procs ← procs + 1
send signal to terminate
```

Algorithm 1: *FarmzeroinNR master* processor pseudocode.

It must be noted that messages exchanged between the *master* and some *worker* in the pipeline need to be routed through the processors that lay in between. For the global performance of the system it is important that messages reach their destination as quickly as possible, therefore communication must be given priority over the computation.

To compute eigenvectors, once we have computed (selected) eigenvalues, we can use inverse iteration. Convergence is fast but eigenvectors associated with close eigenvalues may not be orthogonal. The LAPACK's routine sstein uses re-

```
while not receive signal to terminate do
    interval ← input_channel
    if interval has more than one eigenvalue do
        intervals ← bisection method(interval)
        output ← intervals (to the master)
    else ⟶
        eig ← extract isolate eigenvalue
        output ← eig (to the master)
```

Algorithm 2: *FarmzeroinNR worker* processor pseudocode.

orthogonalization of such eigenvectors. This does not solve the problem when there is a cluster with many close eigenvalues [Dem97, pg. 231], and recent progress on this problem appears to indicate that inverse iteration may be *repaired* to provide accurate, orthogonal eigenvectors without spending more than $O(n)$ flops per eigenvector. This will make bisection, or *zeroinNR* and *repaired* inverse iteration the algorithm of choice in all cases, no matter how many eigenvalues and eigenvectors are desired.

4 Performance Analysis

As already mentioned, a typical architecture for the processor farm model consists of a bidirectional array, forming a single pipeline (SP), with the master placed at one end of the array (Fig. 2).

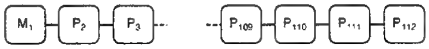


Fig. 2. Single pipeline, with 112 nodes.

It is predictable that as the number of workers increases, the communication overhead becomes more significant and processors that are further away from the master take longer to communicate with him. Furthermore, the activity in the links of the processors which are closer to the master grows with the number of processors and some congestion is to be expected if the computational complexity of each task is not sufficiently large. In an attempt to overcome the problems just mentioned, we decided to test the parallel algorithm with a modified topology, referred to as multiple pipeline (MP), which consists of seven pipelines, each one with 16 transputers; the masters of such pipelines are themselves connected in a single pipeline (Fig. 3).

At the beginning, the interval that contains all the eigenvalues is decomposed in 7 subintervals of equal width which are distributed among the different pipelines.

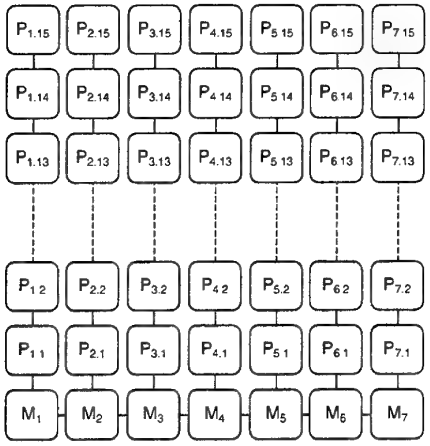


Fig. 3. Multiple pipeline, with 112 nodes.

Although this may reduce to some extent communication overhead and waiting times, it has an important disadvantage which is an eventual deterioration of the load balancing, which becomes critical when some of the subintervals contain a much larger number of eigenvalues than others. Therefore, the spectral distribution of the matrix is an important factor to be considered when comparing the performance of the SP and MP architectures. For this reason we have used four different types of matrices (see Table 1 where $a_i, i = 1, \ldots, n$, represents the diagonal elements $b_i, i = 1, \ldots, n-1$, represents the sub-diagonal elements) with different spectral distributions (see Figures 4, and 5),and sizes n ranging from one thousand to ten thousand.

Matrix	Elements	Analitical Formula
I	$a_i = a$ $b_i = b$	$\left\{ a + 2b\cos\frac{k\pi}{n+1} \right\}_{k=1}^{n}$
II	$a_1 = a - b$ $a_i = a,\ i = 1, \ldots, n$ $a_n = a + b$ $b_i = b,\ i = 1, \ldots, n$	$\left\{ a + 2b\cos\frac{(2k-1)\pi}{2n} \right\}_{k=1}^{n}$
III	$a_i = 0$ $b_i = \sqrt{i(n-i)}$	$\left\{ -n + 2k - 1 \right\}_{k=1}^{n}$
IV	$a_i = -[(2i-1)(n-1) - 2(i-1)^2]$ $b_i = i(n-i)$	$\left\{ -k(k-1) \right\}_{k=1}^{n}$

Table 1. Matrix Types.

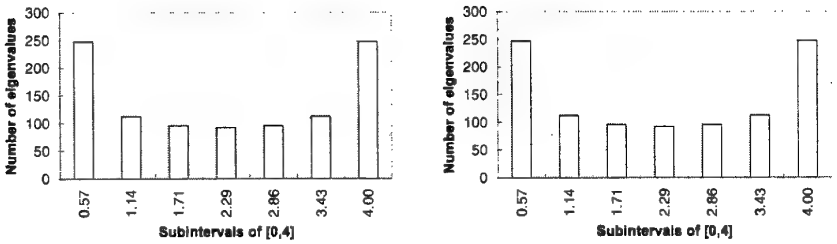


Fig. 4. Spectral distributions for matrix I (left) and matrix II (right), with $n = 1000$.

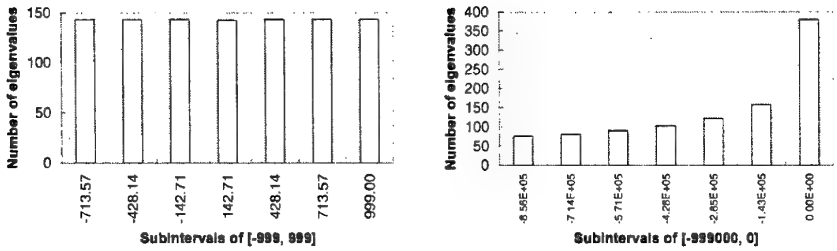


Fig. 5. Spectral distributions for matrix III (left) and matrix IV (right), with $n = 1000$.

We have computed the efficiency in the usual way, i.e.,

$$E = \frac{T_1}{112\ T_{112}}$$

where T_1 represents the time taken by a single transputer executing the sequential implementation of *zeroinNR*, and T_{112} is the time taken by *farmzeroinNR* with 112 processors. In Table 2 such ratios are given, representing by $E(\mathsf{SP})$ and $E(\mathsf{MP})$ the efficiency obtained for the single pipeline and multiple pipeline implementations, respectively.

	Matrix I		Matrix II		Matrix III		Matrix IV	
n	$E(\mathsf{SP})$	$E(\mathsf{MP})$	$E(\mathsf{SP})$	$E(\mathsf{MP})$	$E(\mathsf{SP})$	$E(\mathsf{MP})$	$E(\mathsf{SP})$	$E(\mathsf{MP})$
1000	55%	45%	60%	72%	61%	71%	60%	35%
5000	80%	56%	92%	80%	92%	89%	90%	38%
7000	93%	64%	91%	80%	91%	85%	94%	39%
10000	95%	56%	99%	85%	99%	91%	97%	39%

Table 2. Efficiency of *farmzeroinNR*, for matrices of type I, II, III and IV.

As it can be appreciated from this table, the MP implementation is less efficient than the SP implementation, except for the case of Matrices II and III of

size $n = 1000$. In general, we have obtained better efficiency values with the SP architecture and we conclude that, for n sufficiently large, the communication overhead is not as important as the unbalance in the distribution of tasks introduced by the MP architecture. This is particularly clear in the case of matrix IV since for the larger values of n the efficiency for SP is about 2.4 times better than the efficiency for MP. The explanation for this can be found in Figure 5 (right side): the number of eigenvalues received by each one of the seven pipelines presents, in the case of matrix IV, a large variation, from about 70 to about 370. Another important aspect that must be taken into account is that in the MP implementation there are only 105 *workers*, since 7 processors are playing the role of *master*. However, even if we had used the modified formula

$$E = \frac{T_1}{105\, T_{112}}$$

to compute the efficiency for the MP implementation, the values produced in this way would still be lower than those obtained for the SP architecture in most cases.

5 Conclusions

We have carried out a parallel implementation of an efficient algorithm, dubbed *zeroinNR*, for the eigenvalue problem of a symmetric tridiagonal matrix, on a distributed memory system. The sequential *zeroinNR* method, although not as fast as QR, is consistently faster than simple bisection and retains the excellent numerical properties of this method. We have numerical evidence to support the claim that our method produces eigenvalues with smaller errors than those produced by QR. For the parallel implementation we used a farm model with two different topologies: a single pipeline (SP) of 112 processors and a multiple pipeline implementation (MP) consisting of seven pipelines, each one with 16 processors. The MP architecture reduces the communication overhead to some extent but is not able to retain fully the excellent load balancing of the SP implementation. This trade-off is not clear since it depends on the spectral distribution of each particular matrix. We have used matrices of different types to study this trade-off and conclude that for matrices sufficiently large, the parallel algorithm under the SP architecture performs better than the MP architecture. It must be emphasized that the parallel algorithm under the SP architecture is very efficient: for matrices of size $n = 10000$ we got efficiency values which are in all cases tested larger than 95%.

References

[ABB+95] E. Anderson, Z. Bai, C. Bischop, J. Demmel, S. Hammarling J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and S. Sorensen. *LAPACK User's Guide*. Series: Software, Environments and Tools. SIAM, Philadelphia, PA, 2nd edition edition, 1995.

[BCC+97] L. S. Blackforda, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK User's Guide.* SIAM, Philadelphia, PA, 1997.

[BE78] R. H. Barlow and D. J. Evans. A parallel organization of the bisection algorithm. *The Computer Journal*, (22):267–269, 1978.

[Ber84] H. J. Bernstein. An accelerated bisection method for the calculation of eigenvalue of a symmetric tridiagonal matrix. *Numer. Math.*, (43):153–160, 1984.

[BM+67] W. Barth, R. S. Martin, et al. Calculation of the eigenvalues of a symmetric tridiagonal matrix by the bisection method. *Numer. Math.*, (9):386–393, 1967.

[BW20] A. Baserman and P. Weidner. A parallel algorithm dor determining all eigenvalue of large real symmetric tridiagonal matrices. *Parallel Computing*, (18):1129–1141, 1920.

[Con96] José Manuel Badía Contelles. *Algoritmos Paralelos para el Cálculo de los Valores Propios de Matrices Estructuradas.* PhD thesis, Universidad Politecnica de Valencia, 1996.

[Cup81] J. J. Cuppen. A divide and conquer method for the symmetric eigenvalue problem. *Numer. Math.*, (36):177–195, 1981.

[Dem97] James W. Demmel. *Applied Numerical Linear Algebra.* Society for Industrial and Applied Mathematics, 1997.

[DHvdV93] J. Demmel, M. Heath, and H. van der Vorst. Parallel numerical linear algebra. In *In A. Iserles, Acta Numerica*, volume 2, Cambridge University Press, UK, 1993.

[DS87] J. J. Dongarra and D. C. Sorense. A fully parallel algorithms for the symmetric eigenproblem. *SIAM J. Sci. Stat. Comput*, 8(2):s139–s154, 1987.

[For00] Maria Antónia Forjaz. *Algoritmos Paralelos para o Cálculo de Valores e Vectores Próprios em Sistemas de Multiprocessadores de Memória Distribuída.* PhD thesis, Universidade do Minho, 2000.

[IJ90] I. C. F. Ipsen and E. R. Jessupe. Solving the symmetric tridiagonal eigenvalue problem on the hypercube. *SIAM J. Sci. Stat. Comput*, 11(2):203–229, 1990.

[Kal90] T. Z. Kalambouskis. The symmetric tridiagonal eigenvalue problem on a transputer network. *Parallel Computing*, (15):101–106, 1990.

[LPS87] S. S. Lo, B. Phillipe, and A. Sameh. A multiprocessor algorithm for the symmetric eigenproblem. *SIAM J. Sci. Stat. Comput.* (8):155–165, 1987.

[Mat99] Using matlab. The Math Works Inc., 1999.

[Par80] B. N. Parlett. *The Symmetric Eigenvalue Problem.* Prentice-Hall Series in Computational Mathematics, 1980.

[Ral90] R. Ralha. *Parallel Computation of Eigenvalues and Eigenvectors using Occam and Transputers.* PhD thesis, University of Southampton, 1990.

[Ral93] R. Ralha. Parallel solution of the symmetric tridiagonal eigenvalue problem on a transputer network. In *Proceedings of the Second Congress of Numerical Methods in Engineering,*, Spanish Society of Numerical Methods in Engineering, Spain, 1993.

[RR78] A. Ralston and P. Rabinowitz. *A First Course in Numerical Analysis.* McGraw-Hill, 1978.

[Wil65] J. H. Wilkinson. *The algebraic eigenvalue problem.* Oxford University Press. 1965.

Performance of Automatically Tuned Parallel GMRES(m) Method on Distributed Memory Machines

Hisayasu KURODA[1] *, Takahiro KATAGIRI[1,2], and Yasumasa KANADA[3]

[1] Department of Information Science, Graduate School of Science,
The University of Tokyo
[2] Research Fellow of the Japan Society for the Promotion of Science
[3] Computer Centre Division, Information Technology Center,
The University of Tokyo
2-11-16 Yayoi, Bunkyo-ku, Tokyo 113-8658, JAPAN
Phone: +81-3-5841-2736, FAX: +81-3-3814-2731
{kuroda, katagiri, kanada}@pi.cc.u-tokyo.ac.jp

Abstract. As far as the presently available public parallel libraries are concerned, users have to set parameters, such as a selection of algorithms, an unrolling level, and a method of communication. These parameters not only depend on execution environment or hardware architecture but also on a characteristic of the problems. Our primary goal is to solve two opposite requirements of reducing users parameters and getting high performance. To attain this goal, an auto-tuning mechanism which automatically selects the optimal code is essential. We developed a software library which uses GMRES(m) method on distributed memory machines, the HITACHI SR2201 and HITACHI SR8000. The GMRES(m) method is one of the iterative methods to solve large linear systems of equations. This software library can automatically adjust some parameters and selects the optimal method to find the fastest solution. We show the performance of this software library and we report a case where our library is approximately four times as fast as the PETSc library which is widely used as a parallel linear equation solver.

1 Introduction

Linear algebra, in particular the solution of linear systems of equations and eigenvalue problems, is the basic of general calculations in scientific computing. When a coefficient matrix of linear systems of equations is large and sparse, iterative methods are generally used. For example, if a coefficient matrix is real symmetric and positive definite, the Conjugate Gradient method (CG) is often used. In the case of a nonsymmetric matrix, there are a number of iterative methods with lots of variations [1, 2]. Therefore, in the nonsymmetric case, the most efficient method is left for to further discussion. In addition, there are a few parallel implementations of the iterative methods in the nonsymmetric case. In this

* *Candidate to the Best Student Paper Award*

paper, we focussed on the GMRES(m) which is improved GMRES(Generalized Minimal RESidual method), and developed its library on distributed memory machines.

The GMRES is one of the Krylov subspace solution methods and finds a suitable approximation for the solution x of $Ax = b$ by using the minimum residual approach at every iteration step.

The GMRES(m) restarts the GMRES after each m steps, where m is a suitably chosen integer value. The original method without restarting is often called full-GMRES. This restarting reduces both calculation counts and the size of memory allocation.

This paper is organized as follows. Description of the algorithm of our version of GMRES(m) in Section 2. Section 3 is about the parameters for auto-tuning, and how to search for the optimal parameters. In Section 4, we show auto-tuned parameters and execution time of our library using the auto-tuning methodology. Finally, Section 5 gives conclusions for this paper.

2 The GMRES(m) Algorithm

When given an $n \times n$ real matrix A and a real n-vector b, the problem considered is: Find x which belongs to $\mathbb{R}^n$ such that

$$Ax = b \,. \tag{1}$$

Equation(1) is a linear system. A is the coefficient matrix, b is the right-hand side vector, and x is the vector of unknowns.

Figure 1 shows our version of preconditioned GMRES(m) algorithm. Note that K in the Figure 1 is a preconditioning matrix and this algorithm uses right-preconditioning because of the advantage that it only affects the operator and does not affect the right-hand side.

2.1 Parallel implementation of GMRES(m)

The coefficient matrix A, the vector of unknowns x, the right-hand side vector b, the temporary vectors v_i ($m + 1$ vectors), and orthogonalized vector w are distributed by row-block distribution and each processor element(PE) except for the last PE has the same number of rows. On the other hand, the matrix H, the vector s, and the vector c are the same on every PE.

Since the vector x and v_i are needed to be gathered on every PE, a temporary vector of size n, where n is the size of matrix A, is required.

In the parallel implementation, lines 2, 7, and 29 in the Figure 1 which include matrix-by-vector products and line 9 in the Figure 1 which includes dot product require interprocessor communications.

In lines 14-22 in the Figure 1 which include QR decomposition by using Givens rotations, every PE updates the matrix H which holds the same data. It

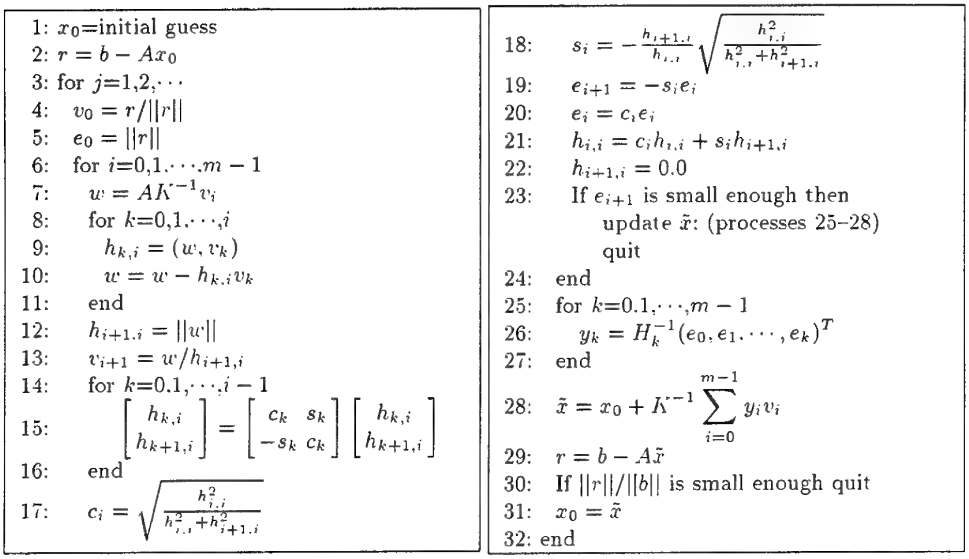


Fig. 1. The preconditioned GMRES(m) algorithm

r, v_i, and w are vectors and if $i \neq j$ then v_i and v_j are different vectors, and not elements of the same vector v. m is the restarting frequency.

seems that holding H is inefficient. However m is several hundreds at the most and the decomposition of the matrix H whose size is $(m+1) \times m$ is inefficient to parallelize. Therefore, the overhead time that every PE updates at the same time is very small.

3 Method for searching parameters

We have developed automatically tuned parallel library by using CG [3]. In this section description of several tuning factors to be considered in preconditioned GMRES(m) algorithm and methods of tuning parameters automatically are provided.

Our library automatically sets several parameters to get high performance. This action is executed after being given a problem. Therefore, our library can select the best method according to a characteristic of the problem. Our library provides a lot of source codes. Users only have to compile them once. While our library code is being executed, the optimal code is selected one after another automatically.

3.1 Matrix storage formats

In the sparse matrix formats, we store the nonzero elements by rows, along with an array of corresponding column numbers and an array of pointers to the beginning of each row (see Figure 2). It is called as *compressed row storage format.*

$$A = \begin{bmatrix} a & b & 0 & 0 \\ c & d & e & 0 \\ 0 & f & g & h \\ 0 & 0 & i & j \end{bmatrix}$$

```
rp[5]={0.2.5.8.10}; /* pointers to the beginning of each row */
cval[10]={0.1.0.1.2.1.2.3.2.3};     /* indices */
aval[10]={a, b, c. d. e, f, g, h, i, j}; /* elements */
```

Fig. 2. Compressed row storage format

In case that the number of nonzero elements at each row are almost equal, compressed row storage format was converted to a matrix format whose size of each row is fixed (see Figure 3). This is called *compressed row storage format for unrolling.* Using such a matrix format, we expected to save the execution time because of the effect of unrolling.

```
cval[12]={0.1.0.0.1.2.1.2.3.2.3.0};      /* indices */
aval[12]={a. b, 0. c. d. e, f, g. h, i. j. 0}: /* elements */
nsize[4]={2.3.3.2}:      /* the number of elements of each row*/
csize=3:                 /* fixed size */
```

Fig. 3. Compressed row storage format for unrolling

Before executing the main iteration, the actual time of the matrix-by-vector product was measured. With this information, we can select the best matrix storage format.

3.2 The stride size of loop unrolling for matrix-by-vector products

To perform the matrix-by-vector product at high performance, we should select the best size of the stride for loop unrolling. This depends on the machine architectures and optimization level of the compilers.

Our library prepares a large number of loop unrolling codes. For example, if the number of nonzero elements of a matrix A at each row is smaller than

10, all expanded loop unrolling codes are examined. In addition to unrolling the inner-loop, we unroll also the outer loop with strides 1, 2, 3, 4, and 8.

There are two types of codes. i.e., a non-prefetch code and a prefetch code. The non-prefetch code uses indirect access just as it is or entrusts the compiler with the treatment. The prefetch code takes off indirect access to the element of arrays (see Figure 4).

Non-prefetch code

```
1: do i=1,10
2:    s = s + a(i) * b(ind(i))
3: end do
```

Prefetch code

```
1: m=ind(1)
2: do i=1,9
3:    s = s + a(i) * b(m)
4:    m=ind(i+1)
5: end do
6: s = s + a(10) * b(m)
```

Fig. 4. Non-prefetch code and prefetch code

As for the prefetch codes, we unroll the outer loop so that the sizes of the stride can be 1, 2, 3, and 4. In total, there are 9 ways of the unrolling codes in each of the number of nonzero elements at each row.

Same as in the case of the matrix format, actual time of the matrix-by-vector product was measured in order to select the best unrolling code in the above.

3.3 How to comunicate in matrix-by-vector products

In matrix-by-vector products, we need a gather operation. Because the elements of a vector are distributed on all of the PEs. We can select the following five implementations for the communications.

No dependence on the location of the nonzero elements of matrix A:

1. Use MPI_Allgather function from the MPI library.
2. Gather in 1 PE then broadcast with MPI_Bcast function.

Dependence on the location of the nonzero elements of matrix A:

3. First use MPI_Isend function, next use MPI_Irecv function.
4. First use MPI_Irecv function, next use MPI_Isend function.
5. Use MPI_Send and MPI_Recv functions.

A communication table is used in the method from 3 to 5. This table indicates the relation between a element index of a vector and a PE number which requires the indexed value. This relation is created from nonzero element indices of a matrix A. By using this communication table, communication traffic becomes very small because an element is transmitted to a PE which requires it. We communicate all of the elements from minimal index to maximal index to other

PEs so that we need only one communication step. In the case that nonzero elements of matrix A are located in limited parts, using the communication table is quite effective.

In the method 5, since both MPI_Send and MPI_Recv functions are blocking communication, execution of other instructions is suspended. However this method saves starting time for the communication. If the total amount of communicated data is small, we can get high performance by selecting the order of communication.

As in the case of the matrix format, the actual time of the matrix-by-vector product was measured and the best way from the alternatives above was selected.

3.4 Restarting frequency

The larger the restarting frequency m, the smaller the iteration count we need. However if m is large, the execution time of one iteration increases with the increment of iteration counts. Because in the orthogonalization, we must calculate a new vector to be orthogonalized to all vectors which have caluculated by earlier iteration (lines 8-11 in the Figure 1). The total amount of calculations for the orthogonalization is proportional to the square of the iteration counts.

There are many ways to decide on m [5]. In our implementation we change the value of m dynamically [6]. Here, let $m_{\max}$ be maximal restarting frequency. We decide on the value of m as follows:

(1) m=2 (initial value).
(2) Add 2 to m if $m < m_{\max}$.
(3) Back to (1).

Our library sets 128 to $m_{\max}$. If the library cannot allocate memory, it sets the maximal size to $m_{\max}$ within the maximum permissible memory allocation. The reason why we decide $m_{\max}$ is to save the amount of calculation for the orthogonalization.

3.5 Gram-Schmidt orthogonalization

Modified Gram-Schmidt orthogonalization (MGS) is often used on single processor machines because of its smaller computational error. However on distributed memory machines, it is not efficient because of the frequent synchronization especially in the case of a large iteration count.

On the other hand, classical Gram-Schmidt orthogonalization (CGS) is efficient on distributed memory machines because the synchronization is needed only once.

In case of using CGS, lines 8–11 in the Figure 1 are replaced as shown in Figure 5.

CGS has less in computational error than MGS. Therefore, our library provides *iterative refinement Gram-Schmidt orthogonalization* [4] (see Figure 6).

```
1:  for k=0,1,···,i
2:    h_{k,i} = (w, v_k)
3:  end
4:  for k=0,1,···,i
5:    w = w − h_{k,i} v_k
6:  end
```

Fig. 5. Classical Gram-Schmidt orthogonalization

```
1:  for k=0,1,···,i
2:    h_{k,i} = (w, v_k)
3:  end
4:  for k=0,1,···,i
5:    w = w − h_{k,i} v_k
6:  end
7:  for k=0,1,···,i
8:    ĥ_k = (w, v_k)
9:    h_{k,i} = h_{k,i} + ĥ_k
10: end
11: for k=0,1,···,i
12:   w = w − ĥ_k v_k
13: end
```

Fig. 6. Iterative refinement Gram-Schmidt orthogonalization

The execution time by this method is twice as large as primary CGS. However it is comparable to MGS in computational error. In our library, we measure the orthogonalization time for calculating a new vector to be orthogonalized to $m_{\max}/2$ vectors. We then select the fastest method, MGS or CGS.

On single processor machines, the MGS is advantageous to the CGS because of localization of memory access. On the other hand, on distributed memory machines, it is not clear which is the best because we must consider the combination of cache memory size, the number of vectors, the number of PEs, interprocessor communications speed and so on.

When the convergence is not improved at two straight steps, we change to the iterative refinement Gram-Schmidt orthogonalization.

3.6 Preconditioning

There are many occasions and applications where iterative methods fail to converge or converge very slowly. Therefore, it is important to apply preconditioning.

In our library, we apply *diagonal scaling* to a coefficient matrix A. In this case, we expect that not only it helps to reduce the condition number and often has a beneficial influence on the convergence behavior but also the computational complexity and memory allocation are reduced by fixing to 1 in all diagonal elements. In addition to the diagonal scaling, we can select the following three implementations.

1. No preconditioning.
2. Polynomial Preconditioning [7].
3. Block incomplete LU decomposition [8].

Let A be the scaled matrix such that $\mathrm{diag}(A) = I$.

In case 2. the matrix A can be written $A = I - B$. and A^{-1} can be evaluated in a Neumann series as

$$A^{-1} = (I - B)^{-1} = I + B + B^2 + B^3 + \cdots . \tag{2}$$

We take a truncated Neumann series as the preconditioner. e.g. approximating A^{-1} by $K^{-1} = I + B$. In this case. K^{-1} is very similar to A but plus and minus signs of elements of K^{-1} are reversed except for the diagonal elements. Since this preconditioning does not need extra memory allocation which holds matrix $I + B$ data. in particular GMRES(m) which requires a lot of memory allocation. it is very useful.

However. approximation of A^{-1} by $I + B$ is efficient only when matrix A is diagonally dominant. namely. spectral radius of B satisfies the relations $\rho(B) < 1$. If $\rho(B) \geq 1$ then $I + B$ does not approximate A^{-1}.

In the preconditioner 3. our library employs zero fill-in ILU factorization called as ILU(0) on each individual block, which is diagonal submatrix on each PE. In this case. we assume $K = LU$.

In lines 7 and 28 in the the Figure 1, there is the matrix-by-vector product in the form of $K^{-1}r$. When we assume $q = (LU)^{-1}r$, we can solve linear system of equations $LUq = r$. where q is the vector of unknowns.

To solve q of the linear system is as follows.

$$\begin{aligned} Lz &= r \quad \text{(Forward substitution)} \\ Uq &= z \quad \text{(Backward substitution).} \end{aligned} \tag{3}$$

where z is a temporary vector. As shown above, it is possible to calculate $K^{-1}r$.

If restarting frequency m is large, the preconditioning is more efficient. Because the overhead time of preconditioning depends on the whole iteration count to converge. setting m at a large value reduces the total iteration count.

The best preconditioning selection is as follows. We iterate the main loop (lines 6–24 in the Figure 1) for $m = 2$ by using every method. Next. we select the method whose relative decrease of the residual norm ($||r_2||/||r_0||$) is the smallest. Note that we do not consider the execution time, we employ the method which reduces the residual norm the most after the same number of iterations.

4 Experimental results

We implemented our auto-tuning methodology on the HITACHI SR2201 and HITACHI SR8000.

The HITACHI SR2201 system is a distributed memory. message-passing parallel machine of the MIMD class. It is composed of 1024 PEs. each having 256

Megabytes of main memory, interconnected via a communication network having the topology of a three-dimensional hyper-crossbar. The peak interprocessor communications bandwidth is 300 Mbytes/s in each direction. We used the HITACHI Optimized Fortran90 V02-06-/D compiler, and compile option we used was *-W0,'opt(o(ss),fold(1))'*. We also used the HITACHI Optimized C compiler. and compile option we used was *+O4 -Wc,-hD1*.

The HITACHI SR8000 system is a distributed memory, message-passing parallel machine of the MIMD class like the HITACHI SR2201. It is composed of 128 nodes. each having 8 Instruction Processors (IPs), 8 Gigabytes of main memory, interconnected via a communication network having the topology of a three-dimensional hyper-crossbar. The peak interprocessor communications bandwidth is 1 Gbytes/s in each direction. We used the HITACHI Optimized Fortran90 V01-00 compiler, and compile option we used was *-W0,'opt(o(4),fold(1))' -noparallel*. We also used the HITACHI Optimized C compiler, and compile option we used was *+O4 -Wc,-hD1 -noparallel*.

We evaluated performance with the following conditions:

- Convergence result : $||r_k||/||r_0|| < 1.0 \times 10^{-12}$
- Initial guess : $x_0 = (0.0, \cdots, 0)^T$
- Precision type : double

4.1 Test problems

Evaluation on our library by employing three problems whose maximal number of nonzero elements of a matrix A at each row is 3. 5, and 7.

Problem 1

The coefficient matrix A is a Toeplitz matrix such as

$$A = \begin{bmatrix} 2 & 1 & & & \\ 0 & 2 & 1 & & \\ R & 0 & 2 & 1 & \\ & R & 0 & 2 & \cdots \\ & & \cdots & \cdots & \end{bmatrix} .$$

where $R = 1.0$, 1.5, and 2.0. The right-hand side vector is $b = (1, 1, \cdots, 1)^T$. The size of matrix A is 4,000.000.

Problem 2

An elliptic boundary value problem of partial differential equation:

$$-u_{xx} - u_{yy} + Ru_x = g(x, y) ,$$
$$u(x, y)|\partial\Omega = 1 + xy ,$$

where the region is $\Omega = [0, 1] \times [0, 1]$, and $R = 1.0$. The right-hand side vector b is set to the exact solution of $u = 1 + xy$. We discretize the region by using a 5-point difference scheme on a 400×400 mesh. The size of matrix A is 160.000.

Problem 3

An elliptic boundary value problem of partial differential equation:

$$-u_{xx} - u_{yy} - u_{zz} + Ru_x = g(x, y, z) ,$$
$$u(x, y)|\partial\Omega = 0.0 .$$

where the region is $\Omega = [0, 1] \times [0, 1] \times [0, 1]$, and $R = 1.0$ and 100.0. The right-hand side vector b is set to the exact solution of $u = e^{xyz} \sin(\pi x) \times \sin(\pi y) \times \sin(\pi z)$. We discretize the region by using a 7-point difference scheme on a $80 \times 80 \times 80$ mesh. The size of matrix A is 512,000.

4.2 The results

Tables 1–3 show the execution time on each problem in the case of no auto-tuning. auto-tuning. and using PETSc[4]. In addition, they show auto-tuned parameters in the auto-tuning case.

The calculation time of the QR decomposition in the lines 14–23 of Figure 1 was less than 1 second on every problem. Even though each PE contains the QR decomposition, this overhead time was very small and it can be ignored.

Following parameters were set as the sample of the no auto-tuning case. These are common parameters which were,used comparison with the no auto-tuning case and the auto-tuning case.

Matrix storage format : Compressed row storage format for unrolling.
Unrolling : Non-prefetch and no unrolling code.
Communication : Use MPI_Allgather funuction from the MPI library.
Restarting frequency : 30 (fixed)
Orthogonalization : Iterative refinement Gram-Schmidt.
Preconditioning : None.

In case of the auto-tuning version the leftmost explanation has the following meaning.

iter. : Iteration count.
time : Total execution time including auto-tuning. (sec)
unro. : Unrolling type. For example, P(2,3) means prefetch code.
two outer loops expanded, and three inner loops expanded.
On the other hand, N(2,3) means non-prefetch code.
com. : Communication type.
Send $\cdots$ use MPI_Send and MPI_Recv in pairs.
Isend $\cdots$ use MPI_Isend and MPI_Irecv in pairs.
Irecv $\cdots$ use MPI_Irecv and MPI_Isend in pairs.
orth. : Orthogonalization type.
prec. : Preconditioning type.
$I + B \cdots$ polynomial preconditioning.
BILU $\cdots$ block incomplete LU decomposition.

The matrix storage format in Tables 1–3 has been omitted since the compressed row storage format for unrolling is selected in all problems.

As for PETSc. we used it with almost default parameter values. For example. the restarting frequency is 30. the technique for orthogonalization is the iterative refinement Gram-Schmidt method and so on. Only the convergence is decided to be set 10^{-12} in order to compare with our library.

Table 1. The results for problem 1

(a) R=1.0 **SR2201**

PE	8	16	32	64	128
No auto-tuning					
iter.	43	43	43	43	43
time	49.6	36.7	25.0	20.9	22.0
Auto-tuning					
iter.	18	19	19	19	20
time	40.5	24.0	14.3	8.3	5.3
unro.	N(1,3)	N(1,3)	N(3,3)	N(3,3)	P(3,3)
com.	Send	Send	Irecv	Irecv	Send
orth.	MGS	MGS	CGS	CGS	CGS
prec.	BILU	BILU	BILU	BILU	BILU
PETSc					
iter.	20	20	21	21	21
time	93.1	45.9	24.5	11.9	6.0

(b) R=1.5 **SR2201**

PE	8	16	32	64	128
No auto-tuning					
iter.	93	93	93	93	93
time	101.2	85.5	56.7	45.7	44.3
Auto-tuning					
iter.	50	93	93	93	93
time.	72.5	29.1	16.6	9.2	5.7
unro.	N(1,3)	N(1,3)	N(3,3)	N(3,3)	N(3,3)
com.	Send	Send	Irecv	Send	Send
orth.	MGS	MGS	CGS	CGS	CGS
prec.	BILU	None	None	None	None
PETSc					
iter.		55	56	57	58
time	fail.	148.0	75.5	38.2	19.7

(c) R=2.0 **SR2201**

PE	8	16	32	64	128
No auto-tuning					
iter.	337	323	323	323	323
time	353.3	277.6	186.9	153.9	143.9
Auto-tuning					
iter.	332	321	321	321	321
time	124.9	86.6	45.0	22.8	13.7
unro.	N(1,3)	N(1,3)	N(3,3)	N(3,3)	P(3,3)
com.	Send	Send	Send	Send	Send
orth.	MGS	MGS	CGS	CGS	CGS
prec.	None	None	None	None	None
PETSc					
iter.					
time	fail.	fail.	fail.	fail.	fail.

(d) R=1.0 **SR8000**

IP	8	16	32	64	128
No auto-tuning					
iter.	43	43	43	43	43
time	25.3	19.0	20.4	20.0	21.0
Auto-tuning					
iter.	18	19	19	19	20
time	23.8	12.9	7.6	5.9	5.1
unro.	N(1,3)	P(2,3)	N(1,3)	N(1.3)	N(1.3)
com.	Send	Send	Send	Send	Irecv
orth.	MGS	MGS	CGS	CGS	CGS
prec.	BILU	BILU	BILU	BILU	BILU

(e) R=1.5 **SR8000**

IP	8	16	32	64	128
No auto-tuning					
iter.	93	93	93	93	93
time	53.7	40.0	43.1	41.8	44.4
Auto-tuning					
iter.	50	93	93	93	93
time	38.7	11.9	7.1	5.7	5.7
unro.	N(2.3)	P(2,3)	N(1,3)	N(1.3)	N(2.3)
com.	Irecv	Send	Send	Send	Send
orth.	MGS	CGS	CGS	CGS	CGS
prec.	BILU	None	None	None	None

(f) R=2.0 **SR8000**

IP	8	16	32	64	128
No auto-tuning					
iter.	323	323	323	323	323
time	180.3	134.8	145.8	141.4	149.7
Auto-tuning					
iter.	321	321	321	321	321
time	52.2	27.3	14.6	9.6	9.2
unro.	N(2.3)	N(2.3)	N(2.3)	N(1.3)	N(1.3)
com.	Irecv	Irecv	Send	Send	Irecv
orth.	MGS	CGS	CGS	CGS	CGS
prec.	None	None	None	None	None

Table 2. The results for problem 2

(a) R=1.0 **SR2201**

PE	8	16	32	64	128
No auto-tuning					
iter.	21842	21842	21842	21842	21842
time	1205.7	769.3	540.9	520.3	413.1
Auto-tuning					
iter.	1349	1328	1429	1596	1497
time	90.7	44.3	24.3	14.1	7.9
unro.	P(3,5)	P(3.5)	P(2.5)	P(2,5)	P(2,5)
com.	Send	Send	Send	Send	Send
orth.	CGS	CGS	CGS	CGS	CGS
prec.	BILU	BILU	BILU	BILU	BILU
PETSc					
iter.	2614	2049	2913	3213	3934
time	576.0	219.3	153.7	81.0	57.0

(b) R=1.0 **SR8000**

IP	8	16	32	64	128
No auto-tuning					
iter.	21842	21842	21842	21842	21842
time	499.7	332.5	278.3	225.8	218.5
Auto-tuning					
iter.	1349	1328	1429	1596	1497
time	45.8	23.6	12.3	7.6	4.5
unro.	P(1,5)	P(1,5)	P(1.5)	P(2.5)	N(1,5)
com.	Send	Send	Send	Send	Send
orth.	CGS	CGS	CGS	CGS	CGS
prec.	BILU	BILU	BILU	BILU	BILU

Table 3. The results for problem 3

(a) R=1.0 **SR2201**

PE	8	16	32	64	128
No auto-tuning					
iter.	1265	1265	1265	1265	1265
time	250.6	149.4	98.9	90.8	80.9
Auto-tuning					
iter.	288	300	417	417	417
time	81.9	42.5	12.5	7.3	4.7
unro.	P(2,7)	P(2,7)	P(1,7)	P(1.7)	P(1,7)
com.	Isend	Isend	Isend	Isend	Isend
orth.	MGS	CGS	CGS	CGS	CGS
prec.	BILU	BILU	$I+B$	$I+B$	$I+B$
PETSc					
iter.	236	254	343	465	538
time	182.2	96.9	67.0	44.2	25.2

(b) R=100.0 **SR2201**

PE	8	16	32	64	128
No auto-tuning					
iter.	626	626	626	626	626
time	124.2	72.1	50.9	44.7	39.3
Auto-tuning					
iter.	176	199	207	306	272
time	45.7	25.3	13.3	11.3	4.3
unro.	P(2,7)	P(1,7)	P(2.7)	P(1.7)	P(1.7)
com.	Isend	Isend	Isend	Irecv	Isend
orth.	MGS	CGS	CGS	CGS	CGS
prec.	BILU	BILU	BILU	BILU	BILU
PETSc					
iter.	203	262	331	310	370
time	156.8	100.6	65.6	29.7	17.1

(c) R=1.0 **SR8801**

IP	8	16	32	64	128
No auto-tuning					
iter.	1265	1265	1265	1265	1265
time	111.4	66.4	42.0	31.5	27.6
Auto-tuning					
iter.	288	300	417	417	417
time	43.9	23.5	6.8	4.8	4.4
unro.	P(1,7)	P(1.7)	P(1,7)	P(1,7)	P(1.7)
com.	Irecv	Isend	Isend	Irecv	Irecv
orth.	CGS	CGS	CGS	CGS	CGS
prec.	BILU	BILU	$I+B$	$I+B$	$I+B$

(d) R=100.0 **SR8801**

IP	8	16	32	64	128
No auto-tuning					
iter.	598	598	598	598	598
time	53.1	31.6	20.0	15.0	13.1
Auto-tuning					
iter.	176	199	207	306	272
time	24.7	14.0	7.8	7.1	3.8
unro.	P(1.7)	P(1.7)	P(1,7)	P(1.7)	P(1.7)
com.	Irecv	Irecv	Isend	Irecv	Irecv
orth.	CGS	CGS	CGS	CGS	CGS
prec.	BILU	BILU	BILU	BILU	BILU

Comparison with no auto-tuning and auto-tuning If the problem size is large, the execution time of auto-tuning is relatively smaller as compared to the total execution time. Tables 1–3 show that auto-tuning method works very well.

Unrolling type In problem 1, non-prefetch code is selected as the unrolling type. In the other problems, prefetch code is selected. In problem 1, our library often selects the code of 3-unrolled outer loop and 3-unrolled inner loop because loop size is small. In problem 3, it often selects no expanded code for the outer loop. These results mean that auto-tuning behavior depends on machine architectures and compilers.

Communication type Since the nonzero elements of a coefficient matrix A was located at near diagonal intensively in all problems, the communication table usage was selected. In the problems 1 and 2, since communication data size was small, the method using MPI_Send and MPI_Recv in pairs was selected so often. In the problem 3, since communication data size was large, the method using MPI_Isend and MPI_Irecv in pairs was selected.

Orthogonalization type If the number of PEs became large, the selected method was changed from the MGS into the CGS. However the number of PEs where the changes happen is different in each problem. For example it changed into the CGS for 32 PEs in problem 1, for 8 PEs in problem 2, and for 16 PEs in problem 3.

Preconditioning type In many cases, BILU was selected as preconditioner. Table 3 shows that $I + B$ is included in the selected method. Because when the number of PEs is large, preconditioning effect of using BILU is small. On the other hand, preconditioning with $I + B$ is invariable and it has nothing to do with the change of the number of PEs.

Comparison to the PETSc In the Tables 1 (b) and 1 (c), the PETSc failed to converge. In this case, users have to set parameters suitably. On the whole, our library is approximately four times as fast as the PETSc library.

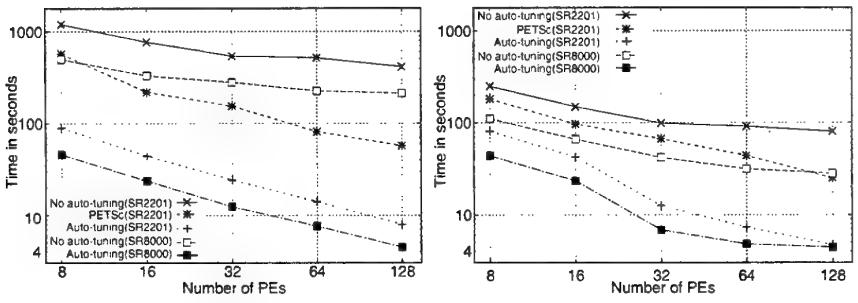
Scalability The execution time is reduced with the number of PEs. Speed-ups for some problems are shown in Figure 7.

5 Conclusion

Selecting optimal codes to get high performance is very important. It brings not only effective utilization of computer resource but also highly user friendly library.

How we can get high performance without setting parameters in detail will be the center of public interest.

Our library is open source and available on-line from out project home page at http://www.hints.org/. Evaluation on the other parallel machines are part of the future work.



(a) Problem 2. R=1.0 (b) Problem 3, R=1.0

Fig. 7. Speed-ups for some problems

Acknowledgments

The authors are much obliged to Dr. Aad van der Steen at the Utrecht University for giving us useful comments in this paper. This research is partly supported by Grant-in-Aid for Scientific Research on Priority Areas "Discovery Science" from the Ministry of Education, Science and Culture, Japan.

References

1. J.J.Dongarra, I.S.Duff, D.C.Sorensen, H.A.van der Vorst: Numerical Linear Algebra for High-Performance Computers, SIAM (1998).
2. Y.Saad: Iterative Methods for Sparse Linear Systems, PWS Publishing Company (1996).
3. H.Kuroda, T.Katagiri, Y.Tsukuda, Y.Kanada: Constructing Automatically Tuned Parallel Numerical Calculation Library — A Case of Symmetric Sparse Linear Equations Solver —, *Proc. 57th National Convention IPSJ*, No.1, pp.1-10 - 1-11 (1998) in Japanese.
4. S.Balay, W.D.Gropp, L.C.McInnes, B.F.Smith: PETSc 2.0 Users Manual, ANL-95/11 - Revision 2.0.24, Argonne National Laboratory (1999).
5. N. Tsuno, T. Nodera: The Speedup of the GMRES(m) Method Using the Early Restarting Procedure, *Trans.IPS.Japan*, Vol.40.No.4,pp.1760–1773 (1998) in Japanese.
6. H. Kuroda, Y. Kanada: Performance of Automatically Tuned Parallel Sparse Linear Equations Solver, *IPSJ SIG Notes* 99-HPC-76-3, pp. 13–18 (1998) in Japanese.
7. O.G.Johnson, C.A.Micchelli, G.Paul: Polynomial Preconditioners for Conjugate Gradient Calculations, *SIAM J. Numer. Anal.*, Vol.20,No.2 (1983).
8. J.S.Kowalik, S.P.Kumar: An Efficient Parallel Block Conjugate Gradient Method for Linear Equations, *Proc.* 1982 *Int. Conf. Par. Proc.*, pp. 47–52 (1982).

A Methodology for Automatically Tuned Parallel Tridiagonalization on Distributed Memory Vector-Parallel Machines

Takahiro Katagiri[12] *, Hisayasu Kuroda[1], and Yasumasa Kanada[3]

[1] Department of Information Science, Graduate School of Science, The University of Tokyo
[2] Research Fellow of the Japan Society for the Promotion of Science
[3] Computer Centre Division, Information Technology Center, The University of Tokyo
2-11-16 Yayoi, Bunkyo-ku, Tokyo 113-8658, JAPAN
Phone: +81-3-5841-2736, FAX: +81-3-3814-2731
{katagiri, kuroda, kanada}@pi.cc.u-tokyo.ac.jp

Abstract. In this paper, we describe an auto-tuning methodology for the parallel tridiagonalization to attain high performance. By searching the optimal set of three parameters for the performance, a highly efficient routine can be obtained automatically. Evaluation of the methodology on the distributed memory parallel machines, the HITACHI SR2201 and HITACHI SR8000, has been provided. The experimental results show 1.3–1.8 times speed-up to a not auto-tuned routine which was specified with reasonable parameters, and the ratios increased for growing problem sizes. Comparison between the execution time of our routine with that of the ScaLAPACK's routine shows that our auto-tuned routine is faster in many cases.

1 Introduction

Tuning computational kernels is time-consuming work. We still have to use several techniques to attain high performance. To avoid the tuning work, many linear algebra programs are constructing by using vendor-tuned BLAS (Basic Linear Algebra Subprograms) routines. The BLAS routines give us high efficiency if the BLAS routines were implemented optimally. However, if the BLAS routines were implemented with low efficiency, the performance will be poor. Solution for such implementation problem for BLAS is to use auto-tuning software for BLAS, such as PHiPAC [1] or ATLAS [11]. We call these software as *auto-tuning software for general usage.*

On the other hand, tuning software automatically that does not or can not use BLAS is hard. Accordingly, every piece of software that can be tuned automatically has a special auto-tuning facility. For example, FFTW [4] for the

* *Candidate to the Best Student Paper Award*

discrete Fourier transformation, and the auto-tuning libraries [9] for sparse linear equation solvers. We call these software packages as *auto-tuning software for dedicated usage*. This paper includes that the report of the development of such auto-tuning software for dedicated usage. The reasons for this report are as follows:

1. Presently, auto-tuning software for parallel processing is not available.
2. We believe that an auto-tuning facility should be contained in each package.

As for reason 2, if the auto-tuning facility is separated from the package, users will be in trouble to attain high performance, because they have to install auto-tuning software into their environments separately. In addition, the time needed for auto-tuning may be enormous because it may tune even non-relevant subroutines (consider the tuning time of all BLAS subroutines.) Hence, our routine contains this auto-tuning facility.

This paper is organized as follows. Description of our parallel dense eigensolver in Section 2. Section 3 is about the parameters of auto-tuning, and how to search for the optimal parameters. In Section 4, we show the results of the auto-tuned parameters and execution time of our routines using the auto-tuning methodology on the HITACHI SR2201 and HITACHI SR8000. The result of the SR2201 includes a comparison with the ScaLAPACK routine. Finally, Section 5 gives the conclusion of this paper.

2 Dense symmetric eigensolver

2.1 Entire process

Our eigensolver can perform the following eigendecomposition:

$$A = X\Lambda X^{-1}, \tag{1}$$

where $A \in \mathbb{R}^{n\times n}$ is a symmetric dense matrix, $\Lambda \in \mathbb{R}^{n\times n}$ is a diagonal matrix which contains eigenvalues $\lambda_i \in \mathbb{R}$, $i = 1, 2, ..., n$ as the i-th diagonal elements, and $X \in \mathbb{R}^{n\times n}$ is a matrix which contains eigenvectors $x_i \in \mathbb{R}^n$ as the i-th row vectors, where n is problem size. In our eigensolver, the decomposition (1) is performed by using a well-known method, the Householder-bisection method. To perform the Householder-bisection method, the following four processes are needed.

1. Tridiagonalization by the Householder method: $T = QAQ$.
2. Eigenvalues of the tridiagonal matrix T are calculated by using the bisection method.
3. By using the inverse-iteration method, eigenvectors of the tridiagonal matrix T are calculated.
   (The processes 2 and 3 yield the eigendecomposition $T = Y\Lambda Y^{-1}$.)
4. Reconstructing eigenvalues for the matrix A : $X = QY$.

Concerning the above four processes, the processes 1 and 4 can affect the whole performance if we need no orthogonalization in process 3. Process 4 depends on the data distribution of the matrices Q and Y [5]. For this reason, determining the optimal parallelization of process 4 is hard, and hence, the parallelization has not been treated in this paper. Next section describes how to parallelize process 1.

2.2 Householder tridiagonalization

Consider the following transformation: $A^{(1)} \equiv A$ to tridiagonal form $A^{(n-2)}$, where $A^{(k)}$ is defined as the k-th iteration of the matrix A. This transformation is denoted by $H^{(k)}(x) = H^{(k)}(A^{(k)}_{k:n,k})$, where $A^{(k)}_{k:n,k}$ is a row vector of A which is constructed by the k-th row and from the k-th to the n-th columns in the k-th iteration. By substituting $H^{(k)} = I - \alpha uu^T$ for $H^{(k)}(x)$ in the $k+1$-th iteration, the following equations are derived:

$$\begin{aligned} A^{(k+1)} &= H^{(k)} A^{(k)} H^{(k)} \\ &= A^{(k)} - \alpha A^{(k)} uu^T - \alpha uu^T A^{(k)} + \alpha^2 uu^T A^{(k)} uu^T \\ &= A^{(k)} - xu^T - uy^T + \alpha uu^T xu^T \\ &= A^{(k)} - uy^T + u\mu u^T - xu^T \\ &= A^{(k)} - u(y^T - \mu u^T) - xu^T, \end{aligned} \tag{2}$$

where

$$x = \alpha A^{(k)} u, \quad y^T = \alpha u^T A^{(k)}, \quad \mu = \alpha u^T x. \tag{3}$$

Here $\alpha, \mu \in \mathbb{R}$, and $u, x, y \in \mathbb{R}^n$. As matrix A is symmetric, $x = y^T$, and we obtain the following formula:

$$A^{(k+1)} = A^{(k)} - u(x^T - \mu u^T) - xu^T. \tag{4}$$

Note that to execute the k-th iteration, the column vector $A_{k:n,k}$ from the partial matrix $A_{k:n,k:n}$ is needed.

2.3 Parallel implementation of the Householder tridiagonalization

Let p be the number of processor elements (PEs). The objective matrix A is distributed by $r \times q$ 2-D grid distribution, called grid-wise distribution (Cyclic, Cyclic), where $r \times q = p$. The grid-wise distribution (Cyclic, Cyclic) distributes the elements of A to the following PEs:

$$a_{ij} \mapsto P_{(i \bmod r,\ j \bmod q)}. \tag{5}$$

where the $P_{(idx,idy)}$. $(idx = 0.1, ..., r-1,\ idy = 0.1, ..., q-1)$ means the PE identification number on the 2-D grid distribution. We do not support block-cyclic distribution because the block-cyclic distribution causes poor load balance when n/p is small.

```
c     P_{myidx,myidy} owns row set Π and
c       column set Γ of n × n matrix A.
⟨1⟩   do k=1, n − 2
⟨2⟩     if (k ∈ Γ) then
⟨3⟩       broadcast(A^(k)_{Π,k}) to
&               PEs sharing rows Π
⟨4⟩     else
⟨5⟩       receive(A^(k)_{Π,k})
⟨6⟩     endif
⟨7⟩     computation of (u_Π, α)
⟨8⟩     if (I have diagonal elements of A)
&       then
⟨9⟩       broadcast(u_Π) to
&               PEs sharing columns Γ
⟨10⟩    else
⟨11⟩      receive(u_Γ)
⟨12⟩    endif
c       computation of x = αA^(k)u
⟨13⟩    do j=k, n
⟨14⟩      if (j ∈ Γ) x_Π = x_Π + α A^(k)_{Π,j} v_j
&         endif
⟨15⟩    enddo
⟨16⟩    global summation of x_Π to PEs
&             sharing rows Π
⟨17⟩    if (I have diagonal elements of A)
&       then
⟨18⟩      broadcast(x_Π) to
&               PEs sharing columns Γ
⟨19⟩    else
⟨20⟩      receive(x_Γ)
⟨21⟩    endif
c       computation of μ = αu^T x
⟨22⟩    do j=k, n
⟨23⟩      μ = αu^T_Π x_Π   enddo
⟨24⟩      global summation of μ to
&               PEs sharing rows Π
c       computation of
c         A^(k+1) = A^(k) − u(x^T − μu^T) − xu^T
⟨25⟩    do j=k, n
⟨26⟩      do i=k, n
⟨27⟩        if (i ∈ Π .and. j ∈ Γ) then
⟨28⟩          update A^(k+1)_{i,j} =
&               A^(k)_{i,j} − v_i (γ^T_j − μv^T_j) − x_i v^T_j
⟨29⟩        endif   enddo   enddo
c       remove k from active columns
c           and rows
⟨30⟩    if (k ∈ Γ)  Γ = Γ − {k}   endif
⟨31⟩    if (k ∈ Π) Π = Π − {k}   endif
⟨32⟩  enddo
```

Fig. 1. Parallel algorithm for the tridiagonalization (the (Cyclic, Cyclic) grid-wise distribution).

We already developed the parallel tridiagonalization and Hessenberg reduction routines [8] by the Householder transformation. Figure 1 shows our parallel tridiagonalization algorithm. The routine of Figure 1 reduces communication and broadcast times for vector reduction to a ratio of $1/\sqrt{p}$. The same idea appears in [3.6.5]. Symmetry of the matrix A was not used in the algorithm of Figure 1. and hence. the algorithm has the computational complexity of $8n^3/3$. while the algorithm using symmetry has $4n^3/3$. This is because. the algorithm based on the symmetry causes complex data accesses. and the complex data accesses prevent easy parallel implementation.

Figure 1 gives a conclusion that implementations of the following three operations affect the total performance.

1. The global summations of the lines ⟨7⟩, ⟨16⟩. and ⟨24⟩.
2. The matrix-vector product of the lines ⟨13⟩–⟨15⟩.
3. The process to update the matrix A of the lines ⟨25⟩–⟨29⟩.

These three operations are the basic operations for parallel tridiagonalization. and the system will tune the three basic operations automatically in our autotuning process.

3 Method for searching parameters

In this section, the method of tuning the three basic operations automatically is described. Hereafter, we use MPI (Message Passing Interface) as the communication library.

3.1 Parameter for the global summations

To perform the global summations, the following two implementations were selected.

1. A routine based on the binary tree-structured communication, or
2. The `MPI_ALLREDUCE` function on MPI.

It depends on the implementation of MPI functions which implementation has the higher performance. Hence, measuring their real performance is necessary to select the best implementation. For that reason, our auto-tuning routine has a parameter for the above two implementations.

3.2 Parameter for the matrix-vector product

To perform the parallel matrix-vector product ($x = \alpha A^{(k)} u$) at high performance, the size of the stride for loop unrolling must be selected. The size of the stride depends on the machine architectures, operating systems, and compilers we use. Therefore, selecting the optimal number of stride without measuring its real execution time is hard.

For example, a three-stride unrolled routine on the matrix-vector product are shown, where the value of `ilocal_length_x` can be divided by 3 to simplify the explanation.

```
m = ilocal_length_x/3
j = 1
do k=1, m
  dt1 = 0.0d0
  dt2 = 0.0d0
  dt3 = 0.0d0
  do i=1, ilocal_length_y
    du_y = u_y(i)
    ix = init_x+i
    iy = init_y+j
    dt1 = dt1 + A(ix, iy  ) * du_y
    dt2 = dt2 + A(ix, iy+1) * du_y
    dt3 = dt3 + A(ix, iy+2) * du_y
  enddo
  x_k(j  ) = dt1 * al
  x_k(j+1) = dt2 * al
  x_k(j+2) = dt3 * al
```

```
  j = j + 3
enddo
```

This example shows a case of the loop unrolling for the outer-loop `k` only. We can unroll the inner-loop `i` or both of the loops `k` and `i`. Current target machines are vector architecture machines as explained in the Section 4. Then, we only unrolled the outer-loop, since unrolling the inner-loop shortens the loop length which is not good for vector architecture machines. For the auto-tuning parameter, we take the size of the stride.

3.3 Parameter for the process to update

As in the case of the matrix-vector product, it is necessary to set the size of the stride for unrolling in the process to update ($A^{(k+1)} = A^{(k)} - u(x^T - \mu u^T) - xu^T$). For example, a two-stride unrolled routine on the process to update is shown, where the value of `ilocal_length_x` also can be divided by 2 to simplify the explanation.

```
m = ilocal_length_x/2
do k=1, m
  j = 2*(k-1)+1
  dtu1 = u_x(j  )
  dtu2 = u_x(j+1)
  dtr1 = mu * dtu1 - x_k(j  )
  dtr2 = mu * dtu2 - x_k(j+1)
  do i=1, ilocal_length_y
    du_y = u_y(i)
    dx_k = x_k(i)
    ix = init_x+i
    iy = init_y+j
    A(ix, iy  ) = A(ix, iy  ) + du_y * dtr1 - dx_k * dtu1
    A(ix, iy+1) = A(ix, iy+1) + du_y * dtr2 - dx_k * dtu2
  enddo
enddo
```

For the same reason as for the matrix-vector product, we only unroll the outer loop `k`. The auto-tuning parameter for the process to update is the stride for unrolling.

3.4 How to search these parameters

Let the parameters for the global summation, the matrix-vector product, and the process to update be denoted as `Comm.Type`, `Mat-Vec`, and `Updating`, respectively. The `Comm.Type` can take on the values { `Tree`, `MPI_ALLREDUCE` }, where `Tree` means a routine based on binary tree-structured communication, and the `MPI_ALLREDUCE` means communication by a MPI function. The `Mat-Vec` can have the values { `None`, 2, 3, 4, 5, 6, 8, 16 }, where the numbers show the size

of the stride for unrolling. The `Updating` can be chosen as { `None`, 2, 3, 4, 5, 6, 8, 16 } like in the `Mat-Vec` case.

Following is the description of how to search for optimal parameters. We first set the following default parameter values:

$$\texttt{Comm.Type} \equiv \texttt{Tree},\ \texttt{Mat-Vec} \equiv 8,\ \texttt{Updating} \equiv 6. \tag{6}$$

Secondly, we searched the optimal parameters by using the above initial parameters. Method for varying the parameters is as follows.

1. `Comm.Type=Tree`, `Mat-Vec=8`, and `Updating` is varied as { `None`, 2, 3, 4, 5, 6, 8, 16 }.
2. `Comm.Type=Tree`, `Mat-Vec` is varied as { `None`, 2, 3, 4, 5, 6, 8, 16 }, and `Updating`={ the selected value from the process 1 }
3. `Comm.Type` is varied as { `Tree`, `MPI_ALLREDUCE` }, `Mat-Vec`={ the selected value from the process 2 }, and `Updating`={ the selected value from the process 1 }.

This method can not find optimal parameters if there is a dependency among the three parameters. However, the basic operations we mentioned are separated physically (see Figure 1), hence, there is no dependency in the three parameters. Therefore, we may be confident that our method can find an almost optimal set of parameters.

As for the problem sizes, $n = 100$ is specified as the initial values. The problem size is increased by using the stride of 100 while problem size n is under 1000, the stride of 1000 while $1000 \leq n < 10000$, and the stride of 10000 while n is over 10000. This increment is used in each searching process.

4 Experimental results

We implemented the auto-tuning methodology on the HITACHI SR2201 and HITACHI SR8000.

The HITACHI SR2201 system is a distributed memory, message-passing parallel machine of the MIMD class. It is composed of 1024 PEs, each having 256 Megabytes of main memory, interconnected via a communication network having the topology of a three-dimensional hyper-crossbar. The peak interprocessor communications bandwidth is 300 Mbytes/s in each direction. We used the HITACHI Optimized Fortran90 V02-06-/D compiler, and the compile option we used was *-rdma -W0,'OPT(O(SS))'*.

The HITACHI SR8000 system is a distributed memory, message-passing parallel machine of the MIMD class like the HITACHI SR2201. It is composed of 128 nodes, each having 8 Instruction Processors (IPs), 8 Gigabytes of main memory, interconnected via a communication network having the topology of a three-dimensional hyper-crossbar. The peak interprocessor communications bandwidth is 1 Gbytes/s in each direction. The SR8000 system has two types

of parallel environments, named inner-node parallel processing and inter-node parallel processing. The inner-node parallel processing is so-called parallel processing in a sheard memory parallel machine, and there is no interprocessor communication. On the other hand, the inter-node parallel processing is like parallel processing as a distributed memory parallel machine, and it can perform interprocessor communications. We used the HITACHI Optimized Fortran90 V01-00 compiler, and compile option we used was *-W0,'OPT(O(SS)).mp(p(0))'* in the inner-node parallel processing, and *-W0,'OPT(O(SS)).mp(p(4))'* in the inter-node parallel processing.

The communication library used for the SR2201 and SR8000 was MPI. Both machines have vector PEs in a sense, i.e. the Pseudo Vector Processor [2]. Therefore, we can regard both machines as vector-parallel machines.

We implemented our tridiagonalization routine by using dedicated subroutines which satisfy functions for the three parameters. For instance, our routine contains a two-stride unrolled matrix-vector product subroutine, or a three-stride unrolled subroutine to update, and so on. By using such subroutines, we can specify the arbitrary parameters. Note that our software does not generate Fortran codes dynamically in this experiments. All auto-tuning was done at run time.

4.1 The results of the SR2201

Results of auto-tuning Table 1 shows parameters auto-tuned on the SR2201. The tuning time depended on the number of PEs, and the CPU elapsed time

Table 1. The auto-tuned parameters on the SR2201.

(a) Case of 4 PEs

Size	Comm.Type	Mat-Vec	Updating
100	MPI_ALLREDUCE	6	3
200	Tree	8	4
300	Tree	8	6
400	Tree	5	2
500	Tree	8	5
600	Tree	5	6
700	Tree	8	6
800	Tree	3	3
900	Tree	8	4
1000	Tree	5	5
2000	Tree	5	6
3000	Tree	5	5
4000	Tree	3	3
5000	MPI_ALLREDUCE	5	5
6000	MPI_ALLREDUCE	5	5
7000	MPI_ALLREDUCE	5	5
8000	MPI_ALLREDUCE	3	2
	Tuning time	118401 [Sec.]	(32.8 [Hours])

(b) Case of 32 PEs

Size	Comm.Type	Mat-Vec	Updating
100	MPI_ALLREDUCE	6	16
200	MPI_ALLREDUCE	4	5
300	MPI_ALLREDUCE	4	4
400	MPI_ALLREDUCE	6	3
500	MPI_ALLREDUCE	6	4
600	MPI_ALLREDUCE	6	4
700	MPI_ALLREDUCE	8	3
800	MPI_ALLREDUCE	5	3
900	MPI_ALLREDUCE	4	3
1000	MPI_ALLREDUCE	5	3
2000	MPI_ALLREDUCE	5	5
3000	MPI_ALLREDUCE	8	5
4000	MPI_ALLREDUCE	5	5
5000	MPI_ALLREDUCE	8	5
6000	MPI_ALLREDUCE	5	5
7000	MPI_ALLREDUCE	5	5
8000	MPI_ALLREDUCE	3	3
	Tuning time	15555 [Sec.]	(4.3 [Hours])

was about 32 hours at most. The tendency of the tuned parameter of `Comm.Type` were different between 4 and 32 PEs, and the tuned parameters of `Mat-Vec` and `Updating` was different on every problem size. From these facts, we expected that the routine is effective in speeding up.

Comparison to ScaLAPACK To evaluate execution time of the tridiagonalization routine (hereafter TRD), we used the HITACHI optimized ScaLAPACK version 1.2 [7]. Its communication library used was PVM, and PBLAS (Parallel BLAS) which is the computational kernel for ScaLAPACK and is optimized by HITACHI limited. ScaLAPACK's tridiagonalization routine (hereafter SLP TRD) is implemented by using block-cyclic distribution, a blocked algorithm, and symmetry of the matrix [10]. Because of using a blocked algorithm, the size of blocking (BL) can greatly affect the performance of ScaLAPACK. According to [7], if the problem size n is less than 4000, the desirable BL is 60, and if n is over 4000, the desirable BL is 100 on the SR2201. Considering these recommended values, we evaluated the performance of the SLP TRD routines with $BL = \{40, 60, 80, 100, 120\}$ to find which BL gives the best performance. In [7] it is shown that $\sqrt{p} \times \sqrt{p}$ is the best layout for the PE grid. We measured execution time in the PE grid for a large number of PEs. When the number of PEs is small, such as 4, 32, and 64, we measured time in all combinations for the PE grid to find which PE grid gives the best performance.

Table 2 shows execution time of the TRD1 (not auto-tuned), TRD2 (auto-tuned), and SLP TRD. Reasonable parameters of `Comm.Type` ≡ `Tree`, `Mat-Vec` ≡ 8, and `Updating` ≡ 6 are specified in the TRD1 (not auto-tuned). Note that the optimal BL size and PE grids for the SLP TRD are used, and the values are included in Table 2.

Table 2. Execution time on the SR2201. Unit is in second.

(a) Case of 4 PEs

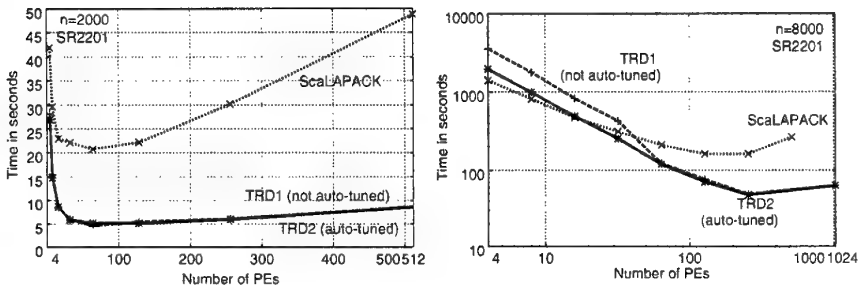
Size	SLP TRD (Grid,BL)	TRD1 (not AT)	TRD2 (AT)	TRD1 /TRD2
100	0.02 (1×4, 100)	0.056 (2×2)	0.056 (2×2)	1.00
200	0.48 (1×4, 100)	0.131 (2×2)	0.133 (2×2)	0.98
400	1.73 (1×4, 40)	0.435 (2×2)	0.475 (2×2)	0.91
800	6.01 (1×4, 40)	3.732 (2×2)	2.454 (2×2)	1.5
1000	9.32 (2×2, 40)	3.817 (2×2)	3.785 (2×2)	1.0
2000	41.90 (2×2, 40)	28.165 (2×2)	26.937 (2×2)	1.0
4000	231.10 (2×2, 40)	411.666 (2×2)	242.010 (2×2)	1.7
8000	1422.69 (2×2, 100)	3589.175 (2×2)	1962.512 (2×2)	1.8

(b) Case of 32 PEs

Size	SLP TRD (Grid,BL)	TRD1 (not AT)	TRD2 (AT)	TRD1 /TRD2
100	0.09 (4×8, 100)	0.108 (4×8)	0.106 (4×8)	1.01
200	0.87 (2×16, 100)	0.250 (4×8)	0.240 (4×8)	1.04
400	2.33 (2×16, 60)	0.514 (4×8)	0.516 (4×8)	0.99
800	6.27 (2×16, 60)	1.207 (4×8)	1.228 (4×8)	0.98
1000	8.28 (2×16, 60)	1.654 (4×8)	1.687 (4×8)	0.98
2000	22.18 (4×8, 40)	5.930 (4×8)	5.886 (4×8)	1.00
4000	72.74 (4×8, 40)	32.961 (4×8)	32.124 (4×8)	1.02
8000	313.25 (4×8, 40)	427.267 (4×8)	254.937 (4×8)	1.6

Table 2 shows that we obtained 1.6–1.8 times speed-ups with respect to the TRD1 (not auto-tuned) when problem sizes were large, such as 4000, 8000. As for the SLP TRD execution time, we find that when problem size is small, the TRD was faster than the SLP TRD. On the other hand, when problem sizes per PE were large, the SLP TRD was faster than the TRD. We consider that this is explained from the computational complexity of the TRD, since the TRD has twice computational complexity to the SLP TRD.

Figure 2 shows the execution time of the TRD1 (not auto-tuned), TRD2 (auto-tuned), and SLP TRD in $n = 2000$ and 8000 cases. Note that the execution time of the SLP TRD in Figure 2 was specified the optimal BL and the PE grid. From Figure 2, we can conclude that when $n = 2000$, the TRD is always faster



(a) Case of $n = 2000$ (b) Case of $n = 8000$

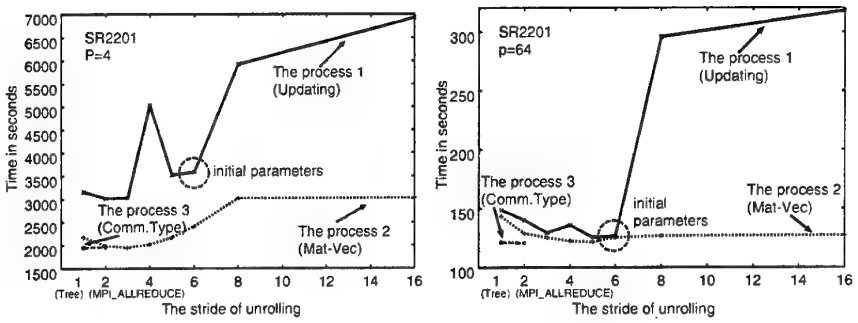
Fig. 2. Execution time for the SLP TRD and TRD in the tridiagonalization (SR2201).

than the SLP TRD, and the speed-up ratios are about 2–6 times. On the other hand, when $n = 8000$, the execution speed of the TRD was slower than the SLP TRD when the number of PEs was under 64, however, when over 64, the TRDs became faster than the SLP TRD. The effect of auto-tuning was high when the number of PEs was under 64.

From the experimental results, we conclude that our methodology is useful, especially, when the problem sizes are large. In addition, the TRD is fast when the problem sizes are small on the SR2201.

The execution time in every auto-tuning process To evaluate the auto-tuning process in detail, we analyzed the execution time in each of our auto-tuning process. Figure 3 shows the time when the problem size was 8000.

From Figure 3 (a), we see that the specified initial parameters (`Comm.Type` ≡ `Tree`, `Mat-Vec` ≡ 8, `Updating` ≡ 6) were worse than the case of Figure 3 (b), because the 6-stride of the process 1 (`Updating`) and the 8-stride of the process 2 (`Mat-Vec`) in Figure 3 (a) were not optimal parameters, and the change of the elapsed time when varying these strides was high. Hence, we conclude that the initial parameters were not good for the case of 4 PEs, and this caused



(a) Case of $p = 4$ (b) Case of $p = 64$

Fig. 3. The execution time in every auto-tuning process. (SR2201, $n = 8000$)

high speed-up ratios. On the other hand, Figure 3 (b) shows that the initial parameters we specified were almost optimal values. For this reason, we conclude that we did not obtain better speed-ups on 64 PEs than the speed-ups on 4PEs on the SR2201.

4.2 The results of SR8000

Results of auto-tuning and execution time Table 3 shows auto-tuned parameters on the SR8000. From Table 3, the tendency of tuned parameters was found to be different between the inner-node parallel and inter-node parallel environments. From this fact, we could also find the cases for the speed up. Table 4 shows execution time of the TRD1 (not auto-tuned) and TRD2 (auto-tuned). From Table 4, we obtained about 1.1–1.3 times speed-ups with respect to the TRD1 (not auto-tuned). The effect became stronger when problem size increased. So, the authors conclude that our auto-tuning methodology is also useful on the SR8000.

5 Conclusion

The authors have implemented and evaluated a tridiagonalization routine by using an auto-tuning methodology. Selecting suitable implementations for the global summation, the matrix-vector product, and the process to update on the parallel tridiagonalization is the auto-tuning methodology we mentioned in this paper, and the methodology is quite simple. Even though we used this quite simple methodology, we could obtain about 1.1–1.8 times speed-ups with respect to the routine for which the reasonable parameters in the SR2201 and the SR8000 were specified. From these results, the authors concluded that such an auto-tuning methodology is an effective technique.

The auto-tuning methodology is for vector-parallel machines. The auto-tuning methodology for the RISC based parallel machines, such as selecting blocking factors in blocked algorithms, and evaluation on the RISC based parallel machines are parts of the future work.

Table 3. The auto-tuned parameters on the SR8000.

(a) Case of 1 Node (8 IPs) (SR8000, inner-node parallel, sheard memory)

Size	Comm.Type	Mat-Vec Updating	
100	MPI_ALLREDUCE	None	None
200	MPI_ALLREDUCE	4	None
300	MPI_ALLREDUCE	8	None
400	MPI_ALLREDUCE	4	None
500	MPI_ALLREDUCE	5	None
600	MPI_ALLREDUCE	6	3
700	MPI_ALLREDUCE	6	None
800	MPI_ALLREDUCE	6	3
900	MPI_ALLREDUCE	6	None
1000	MPI_ALLREDUCE	6	3
2000	MPI_ALLREDUCE	6	None
3000	MPI_ALLREDUCE	6	None
4000	MPI_ALLREDUCE	6	None
5000	MPI_ALLREDUCE	4	None
6000	MPI_ALLREDUCE	4	None
7000	MPI_ALLREDUCE	6	None
8000	MPI_ALLREDUCE	6	None
Tuning time		16325 [Sec.]	(4.5 [Hours])

(b) Case of 4 Nodes (32 IPs) (SR8000, inter-node parallel, distributed memory)

Size	Comm.Type	Mat-Vec Updating	
100	Tree	None	2
200	Tree	None	None
300	Tree	None	2
400	Tree	None	None
500	Tree	None	None
600	Tree	None	None
700	Tree	None	None
800	Tree	4	None
900	Tree	4	None
1000	Tree	6	None
2000	MPI_ALLREDUCE	6	4
3000	MPI_ALLREDUCE	6	4
4000	MPI_ALLREDUCE	4	16
5000	MPI_ALLREDUCE	4	16
6000	MPI_ALLREDUCE	4	16
7000	MPI_ALLREDUCE	6	16
8000	MPI_ALLREDUCE	6	16
Tuning time		4443 [Sec.]	(1.2 [Hours])

Table 4. Execution time on the SR8000. Unit is in second.

(a) Case of 1 Node (8 IPs) (SR8000, inner-node parallel, sheard memory)

size	TRD1 (not AT)	TRD2 (AT)	TRD1 /TRD2
100	0.024 (2×4)	0.022 (2×4)	1.09
200	0.053 (2×4)	0.049 (2×4)	1.08
400	0.162 (2×4)	0.145 (2×4)	1.11
800	0.678 (2×4)	0.587 (2×4)	1.15
1000	1.155 (2×4)	0.988 (2×4)	1.16
2000	7.098 (2×4)	5.595 (2×4)	1.26
4000	50.451 (2×4)	39.263 (2×4)	1.28
8000	389.297 (2×4)	308.307 (2×4)	1.26

(b) Case of 4 Nodes (32 IPs) (SR8000, inter-node parallel, distributed memory)

Size	TRD1 (not AT)	TRD2 (AT)	TRD1 /TRD2
100	0.038 (2×2)	0.036 (2×2)	1.05
200	0.077 (2×2)	0.072 (2×2)	1.06
400	0.176 (2×2)	0.162 (2×2)	1.08
800	0.490 (2×2)	0.450 (2×2)	1.08
1000	0.714 (2×2)	0.648 (2×2)	1.10
2000	2.806 (2×2)	2.345 (2×2)	1.19
4000	14.957 (2×2)	11.392 (2×2)	1.31
8000	102.369 (2×2)	75.398 (2×2)	1.35

Acknowledgments

The authors are much obliged to Dr. Aad van der Steen at the Utrecht University for giving us useful comments in this paper. This research is partly supported by Grant-in-Aid for Scientific Research on Priority Areas "Discovery Science" from the Ministry of Education, Science and Culture, Japan.

References

1. J. Bilmes, K. Asanović, C.-W. Chin, and J. Demmel. Optimizing Matrix Multiply Using PHiPAC: A Portable, High-performance, ANSI C Coding Methodology. In *Proceedings of International Conference on Supercomputing 97* (Vienna, Austria, 1997) 340–347.
2. T. Boku, K. Itakura, H. Nakamura, and K. Nakazawa. CP-PACS: A Massively Parallel Processor for Large Scale Scientific Calculations. In *Proceedings of International Conference on Supercomputing 97* (Vienna, Austria, 1997) 108–115.
3. H. Chang, S. Utku, M. Sakama, and D. Rapp. A Parallel Householder Tridiagonalization Stratagem Using Scattered Square Decomposition. *Parallel Computing* **6** (1988) 297–311.
4. M. Frigo. A Fast Fourier Transform Compiler. In *Proceedings of the 1999 ACM SIGPLAN Conference on Programming Language Design and Implementation* (Atlanta, Georgia, 1999) 169–180.
5. B. Hendrickson, E. Jessup, and C. Smith. Toward an Efficient Parallel Eigensolver for Dense Symmetric Matrices. *SIAM J. Sci. Comput.* **20(3)** (1999) 1132–1154.
6. B. A. Hendrickson and D. E. Womble. The Tours-wrap Mapping for Dense Matrix Calculation on Massively Parallel Computers. *SIAM Sci. Comput.* **15(5)** (1994) 1201–1226.
7. HITACHI Ltd. Using ScaLAPACK and PBLAS Libraries for the HITACHI SR2201. *Computer Centre News, the University of Tokyo* **30(2)** (1998) 36–58. in Japanese.
8. T. Katagiri and Y. Kanada. Performance Evaluation of Householder Method for the Eigenvalue Problem on Distributed Memory Architecture Parallel Machine. *IPSJ SIG Notes 96-HPC-62* (1996) 111–116. in Japanese.
9. H. Kuroda and Y. Kanada. Performance of Automatically Tuned Parallel Sparse Linear Equations Solver. *IPSJ SIG Notes, 99-HPC-76* (1999) 13–18. in Japanese.
10. K. S. Stanley. *Execution Time of Symmetric Eigensolver*. Ph.D Thesis, The University of California at Berkeley, 1997.
11. R. C. Whaley and J. J. Dongarra. Automatically Tuned Linear Algebra Software. ATLAS project, http://www.netlib.org/atlas/index.html.

A new Parallel approach to the Toeplitz Inverse Eigenproblem using Newton-like Methods.

Jesús Peinado[1], Antonio M. Vidal[2]

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia. Valencia, 46071, Spain
[1]jpeinado@dsic.upv.es (Author in charge of correspondence),
[2]avidal@dsic.upv.es
Phone :+(34)-6-3877798. Fax:+(34)-6-3877359

Abstract. In this work we show several portable sequential and parallel algorithms for solving the inverse eigenproblem for Real Symmetric Toeplitz matrices. The algorithms are based on Newton's method (and some variations), for solving nonlinear systems. We exploit the structure and some properties of Toeplitz matrices to reduce the cost, and use Finite Difference techniques to approximate the Jacobian matrix. With this approach, the storage cost is considerably reduced, compared with parallel algorithms proposed by other authors. Furthermore all the algorithms are efficient in computational cost terms. We have implemented the parallel algorithms using the parallel numerical linear algebra library SCALAPACK based on the MPI environment. Experimental results have been obtained using two different architectures: a shared memory multiprocessor, the SGI PowerChallenge, and a cluster of Pentium II PC's connected through a Myrinet network. The algorithms obtained show a good scalability in most cases.

1 Introduction and objectives

In this work we show several portable sequential and parallel algorithms for solving the inverse eigenproblem for Real Symmetric Toeplitz (RST) matrices. These matrices appear in several numerical problems in physics and engineering. There are many

references related to solving Toeplitz linear systems, however references related to the Toeplitz inverse problem are limited. Parallel computing is specially appropriate due to the high computational cost of solving this problem.
The algorithms presented in this paper are based on Newton's method, (Newton, Shamanskii and Chord methods, and the Armijo Rule) [9], for solving large scale general nonlinear systems. We exploit the structure and some properties of the Toeplitz matrices to reduce the cost. We use finite difference techniques [9] to approximate the Jacobian Matrix. Our idea is to use as standard a method as possible.
Our approach to solve the problem as a general nonlinear system is different from other "state of the art" sequential [18] and parallel [3] algorithms. Our algorithms considerably reduce storage cost, and thus allow us to work with larger problems. Furthermore, our algorithms are efficient in computational terms.
To carry out the experimental study, we worked with 15 test problems detailed in [2][15]. Each problem has a different pattern (spectrum) of eigenvalues. To compare with other nonlinear system methods we used Powell's method, implemented in the MINPACK-1 [14] standard package. Powell's method is a robust general purpose method to solve nonlinear systems.
We implemented all the algorithms using portable standard packages. In the case of sequential algorithms we used LAPACK [1] numerical linear algebra library. And for parallel algorithms we used SCALAPACK [4] and BLACS [19] libraries, based on the parallel MPI [3] environment. All programs have been implemented using C++ language.
Experimental results have been obtained using two different architectures: a shared memory multiprocessor, the SGI PowerChallenge, and a cluster of Pentium II PC's connected through a Myrinet [13] network. However other machines could be used due to the portability of the packages and our code.
In both machines we obtained good results and all the algorithms are scalable. The scalability of the algorithms is specially good even when working with problems where the initial size of the scalability test is small.
We want to emphasize the behaviour of the algorithms using the cluster of PC's and the Myrinet network. This system is a good, cheap alternative to more expensive systems because the ratio performance/price is higher than when using classical MPP machines.

2 The problem

Let $t = \{t_0, t_1, \ldots, t_{n-1}\}$ where $t_0, t_1, \ldots, t_{n-1}$ are real numbers, and let $T(t)$ be the real symmetric Toeplitz (RST) matrix:

$$T(t) = \left(t_{|i-j|}\right)_{i,j=1}^{n} .$$

We say that t generates $T(t)$, and denote the eigenvalues of $T(t)$ by:

$$\lambda_1(t) \le \lambda_2(t) \le \ldots \le \lambda_n(t) .$$

The inverse problem [18] for RST matrices can be described as follows:

Given n real numbers $\hat{\lambda}_1 \le \hat{\lambda}_2 \le \ldots \le \hat{\lambda}_n$, find an n-vector $\hat{t}$ such that:

$$\lambda_i(\hat{t}) = \hat{\lambda}_i, 1 \le i \le n .$$

We will call $\hat{\lambda}_1 \le \hat{\lambda}_2 \le \ldots \le \hat{\lambda}_n$ *target eigenvalues*, and $\hat{\Lambda} = \left[\hat{\lambda}_1, \hat{\lambda}_2, \ldots, \hat{\lambda}_n\right]^T$ *the target spectrum*.

We will use nonlinear system techniques to find $\hat{t} = \left[\hat{t}_0, \hat{t}_1, \ldots, \hat{t}_{n-1}\right]^T$ where $\hat{t}_0, \hat{t}_1, \ldots, \hat{t}_{n-1}$ are the RST matrix coefficients. Thus, $\hat{t}^{(0)}$ is the starting point and we construct an iterative process which converges to $\hat{t}^{(i)} = \left[\hat{t}_0^{(i)}, \hat{t}_1^{(i)}, \ldots, \hat{t}^{(i)}_{n-1}\right]^T$. The n-vector $\hat{t}^{(i)}$ must fulfill $T(\hat{t}^{(i)})$ eigenvalues are the *target spectrum* eigenvalues.

2.1 Newton's Method

We used Newton's method (and some variations) to solve our nonlinear system, because this method is powerful and it has quadratic local convergence [9]. Newton's method [9] is based on the following algorithm, where J is the Jacobian Matrix and F is the function (k is the iteration number):

$$J(x^k)p^k = -F(x^k), \qquad J(x^k) \in \Re^{n \times m},$$
$$x^{k+1} = x^k + p^k \qquad p^k,\ F(x^k) \in \Re^n .$$

We also used several variations to Newton's method: if we only compute the Jacobian matrix and factorize in the first iteration, we use the Chord method. If we compute the Jacobian matrix and factorize in some iterations, this is the Shamanskii method. These changes reduce the time cost of Newton's method, because far fewer Jacobian evaluations and factorizations are performed, however convergence is q-linear [9]. This can be showed better if we write the transition from x^k to x^{k+1}:

$$y_1 = x^k - J(x^k)^{-1} F(x^k)$$
$$y_{j+1} = y_j - J(x^k)^{-1} F(x^k) \quad \text{for } 1 \le j \le m-1,$$
$$x^{k+1} = y_m .$$

Note that $m = 1$ is Newton's method and $m = \infty$ is the Chord method. Other values of m define the Shamanskii method. These methods are frequently used for very large problems.
To improve convergence, we used the Armijo rule. The idea is to convert the above methods from local to global convergence [9]. This improvement allows us to reach convergence in some cases.

2.2 Adapting Newton's method to the inverse Toeplitz eigenproblem

Newton's method must be characterized for the problem to be solved. We must characterize each step to the inverse Toeplitz eigenproblem:

The starting point (x^0): we used two different starting points. Both of them are experimental and depend on the problem to solve (the target spectrum).

Normalized [3] Laurie[11]:

$$t_i = \begin{cases} \dfrac{1}{\sqrt{2(n-1)}} & \text{if } i = 1 \\ 0 & \text{if } i \neq 1 \end{cases}$$

or Trench[18]:

$$t_i = \begin{cases} \dfrac{1}{M_n i^2} & \text{if } i \text{ is odd} \\ 0 & \text{if } i \text{ is even} \end{cases} \quad \text{with } M_n = \sqrt{\sum_{j=1}^{n-1} \left[1 + (-1)^{j+1}\right] \left(\frac{n-j}{j^4}\right)}.$$

The function $F(x^k)$: is the value of the function at the k iteration.

Let t be the vector that generates $T(t)$.
Let $\hat{\Lambda} = [\hat{\lambda}_1, \hat{\lambda}_2, \ldots, \hat{\lambda}_n]$ the target spectrum, then $F(x)$, $x \in \Re^n$, is defined as $F(x) = eig(T(x)) - \hat{\Lambda}$.

where

$$eig(T(x)) = [\lambda_1(x), \lambda_2(x), \ldots, \lambda_n(x)]^T.$$

The computational cost of computing the eigenvalues of a large matrix is high. We can exploit here some of the properties of the Toeplitz matrices. If we use Cantoni and Butler's theorems [5], we can obtain the eigenvalues of the matrix from the eigenvalues of two matrices half its size.
The Jacobian matrix $J(x^k)$: is the value of the Jacobian matrix at the k iteration. We must compute it with the forward difference approximation technique:

$$\frac{\partial f_j}{\partial x_k}(x^{(i)}) \approx \frac{f_j(x^{(i)} + e_k h) - f_j(x^{(i)})}{h}.$$

This technique produces a great increase in the time cost, because $F(x)$ has to be computed once per iteration, and, $F(x + he_j)$, $j = 1,2...n$ must be computed once per column j of the Jacobian matrix. This is the most time consuming step in our algorithms. However, this cost can be alleviated slightly as the entries of the first column do not have to be computed because they are 1:

$$\frac{eig(T(t^{(i)} + he_1)) - eig(T(t^{(i)}))}{h} = \frac{eig(T(t^{(i)})) + h - eig(T(t^{(i)}))}{h} = \frac{h}{h} = 1.$$

Alternative techniques to construct the Jacobian matrix can be found in [11] and [18], but those techniques imply the construction and storage of the eigenvectors of an RST matrix. The use of the difference approximation alleviates the computational and storage cost of this step.

Storage cost is determined by the costs of computing the eigenvalues, and solving a linear system. The cost is the same for the three methods: 1) Storing the matrices to compute F, and 2) Storing the Jacobian matrix and the linear system. The cost of 1 consists of storing two half sized problem matrices, and the cost of 2 is storing one problem sized matrix.

The linear system: the Jacobian matrix has no special structure or property. We used the LU solver and forward and backward substitution for solving two triangular systems [8].

2.3 The sequential algorithm

To carry out the sequential algorithm we used all the techniques explained in the former section. Furthermore to obtain a code as efficient as possible we used the linear algebra library LAPACK. The Newton algorithm particularized for the inverse Toeplitz eigenproblem will be as follows:

The Newton sequential Algorithm (for the Inv. Toeplitz eigenproblem)

Choose a starting point x^0

Compute vector $F(x^k)$:

$v \leftarrow F(x^k) = eig(T(x^k))$ (* $F(x^k) + \Lambda$ *)

While the stopping criterion has not been reached

Compute Jacobian Matrix $J(x^k)$:
  If column j=1
    $J(x^k)_1 = [1,1...1]^T$
  else
    For j=2:n
      $w \leftarrow F(x^k + he_j) = eig(T(x^k + he_j))$ $(* F(x^k + he_j) + \Lambda *)$
      $J(x^k)_j = \dfrac{w - v}{h}$
Solve the linear system $J(x^k)s^k = -\left(F(x^k) + \Lambda\right)$:
  Factorize $J(x^k) = LU$
  Solve $LUs^k = -\left(F(x^k) + \Lambda\right)$ $(* \ F(x^k) + \Lambda \ *)$
Update the iterate $x^{k+1} = x^k + s^k$
Compute vector $F(x^k)$:
  $v \leftarrow F(x^k) = eig(T(x^k))$ $(* \ F(x^k) + \Lambda \ *)$

The Chord and Shamanskii Methods have similar algorithms.

The computational cost of the sequential algorithm will depend on the iteration cost. The algorithm uses routines to: compute the eigenvalues, add two vectors, solve a linear system, compute vector norms and merge two vectors. The final expression is as follows:

Newton	**Chord**	**Shamanskii**
$k_N\left(\frac{n^4}{3} + \frac{2n^3}{3}\right)$	$\frac{n^4}{3} + k_c\frac{n^3}{3}$	$k_s\left(\frac{n^4}{3} + m\frac{n^3}{3}\right)$.

where k_N, k_s and k_c are the respective iterations for Newton's, the Shamanskii and Chord methods. A more detailed analysis can be found in [15].

Storage cost is the same for the three methods: 1) Storing the matrices to compute F, and 2) Jacobian matrix and the linear system. The cost of 1 consists of storing two half sized problem matrices, and the cost of 2 is storing one problem sized matrix:

$$\text{Cost} = 2\left(\frac{n}{2}\right)^2 + n^2 = \frac{3n^2}{2} .$$

3 Parallel algorithm

3.1 How to parallelize the sequential algorithm

To carry out this work, we worked with the SCALAPACK package. We parallelized the computation of the Jacobian, the solution of linear system, and the updating of the iterate. All the steps are parallelized to iteration level.

SCALAPACK algorithms work with a 2-D logical mesh and a bidimensional block cyclic data distribution. Below is a possible distribution of a 9 x 9 Jacobian Matrix, before computing the LU decomposition, for the case of a 2 x 3 mesh of processors:

		0			1			2	
		j_{11}	j_{12}	j_{17}	j_{18}	j_{13}	j_{14}	j_{19}	j_{15}	j_{16}
		j_{21}	j_{22}	j_{27}	j_{28}	j_{23}	j_{24}	j_{29}	j_{25}	j_{26}
0		j_{51}	j_{52}	j_{57}	j_{58}	j_{53}	j_{54}	j_{59}	j_{55}	j_{56}
		j_{61}	j_{62}	j_{67}	j_{68}	j_{63}	j_{64}	j_{69}	j_{65}	j_{66}
		j_{91}	j_{92}	j_{97}	j_{98}	j_{93}	j_{94}	j_{99}	j_{95}	j_{96}
		j_{31}	j_{32}	j_{37}	j_{38}	j_{33}	j_{34}	j_{39}	j_{35}	j_{36}
		j_{41}	j_{42}	j_{47}	j_{48}	j_{43}	j_{44}	j_{49}	j_{45}	j_{46}
1		j_{71}	j_{72}	j_{77}	j_{78}	j_{73}	j_{74}	j_{79}	j_{75}	j_{76}
		j_{81}	j_{82}	j_{87}	j_{88}	j_{83}	j_{84}	j_{89}	j_{85}	j_{86}

	0	1	2
0	$-F_{11}$ $-F_{21}$ $-F_{51}$ $-F_{61}$ $-F_{91}$		
1	$-F_{31}$ $-F_{41}$ $-F_{71}$ $-F_{81}$		

Fig. 1. SCALAPACK block cyclic distribution for the LU algorithm and the right hand side.

A standard SCALAPACK distribution could be as follows: distributing a matrix of *M* rows x *N* columns, partitioned in *MB* x *NB* sized blocks on a 2-D processor mesh with *P* processors. The mesh size is P_r row processors by P_c column processors. The *(i,j)* entry is located on the processor (p_r, p_c) as follows:

$$(p_r, p_c) = [((i-1) \text{ div } MB) \bmod P_r\ ,\ ((j-1) \text{ div } NB) \bmod P_c]$$
$$0 \le p_r \le P_r - 1, \quad 0 \le p_c \le P_c - 1,$$
$$1 \le i \le M, \quad 1 \le j \le N.$$

Analogously on (p_r, p_c) the entries *(i,j)* are located:

$$(i, j) = (x * MB * P_r + p_r * MB + k \ \ , \ \ y * NB * P_c + p_c * NB + l \ )$$

$$x = 0 \ldots \left(\frac{M}{MB * P_r} \right) - 1 \qquad k = 1 \ldots MB$$

$$y = 0 \ldots \left(\frac{N}{NB * P_c} \right) - 1 \qquad l = 1 \ldots NB \ .$$

For the right hand side vector we must reference the indexes showing the row entries.

We oriented our algorithms to optimize the computation of Jacobian matrix *J*, because it is the most time consuming step. First, we need to apply the forward difference approximation formula to compute the Jacobian matrix. To do this $F(x)$ must be replicated in each processor. Therefore F is always computed sequentially in each processor. As we obtained $F(x)$ before, we only need to compute $F(x + he_j)$ with $j = 1,2 \ldots n$. Our suggestion for performing this computation efficiently is the following:

Each column of processors in the logical mesh provided by the SCALAPACK package, is in charge of computing a set of columns in the Jacobian Matrix. For example Fig. 2 shows that, processors (p_x, p_0) must compute the columns 1,2,7,8, processors (p_x, p_1) must compute the columns 3,4,9 and so on.

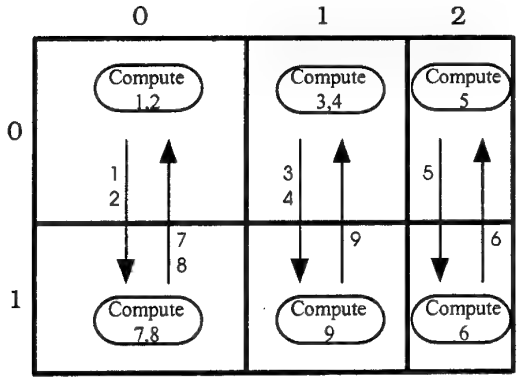


Fig. 2. Computing the Jacobian in parallel.

In addition, the work corresponding to a column of processors is divided among the processors in that column. Thus, in our example processor (p_0, p_0) computes columns 1,2 and processor (p_1, p_0)computes columns 7,8. The same idea is applied to all the processors. Finally, a local communication between processors must be carried out in the same column, in order to achieve the adequate ditribution of the Jacobian matrix.

Once the Jacobian Matrix has been computed, we need to solve the linear system and update the iterate. For solving the linear system, we used the `PDGESV` (and some variations) SCALAPACK routine. The distribution of the elements is shown in Fig. 1.

When the system is solved the solution s is left on the right hand side vector. Then we only have to broadcast s to all the processors, and update the iterate.

For broadcasting s, we used two BLACS steps:

1. All the first column processors must have the complete $s^{(k)}$.
2. Each first column processor sends the complete vector to the processors located in its same row.

Each step can be performed by calling one routine in BLACS.

With these steps the vector is located in all the processors, and we only have to update $x^{(k+1)} = x^{(k)} + s^{(k)}$.

From the point of view of a processor that belongs to the mesh of (P_r, P_c) processors, the algorithm must be as follows:

The Newton Parallel Algorithm (for the Inv. Toeplitz eigenproblem)

Choose x^0 (* same on all the processors *)
Repeat

Compute F using:
$F(x) = eig(T(x))$ (* $F(x^k) + \Lambda$ *)

Compute Jacobian Matrix $J(x^k)$:
For the (n/P_r) columns of (P_r, P_c)
If column $j = 1 \in (p_r, p_c)$
$J(x^k)_1 = [1,1,\ldots,1]$
else
$w \leftarrow F(x^k + he_j) = eig(T(x^k + he_j))$ (* $F(x^k + he_j) + \Lambda$ *)
$J(x^k)_j = \frac{w - v}{h}$
Exchange the (n/P_r) rows with the rows belonging to the processors (i, p_c) $i = 0 \ldots P_r - 1$, $i \neq p_r$

Solve the linear system $J(x^k)s_k = -F(x^k)$:
using the SCALAPACK'S pdgesv$(J(x^k), -F(x^k), s, p_r, p_c)$

```
  Update the iterate:
    If (p_c = 0) ( * column 0 processors *)
      Update the subvector x adding the subvector s
      Broadcast the subvector x
    else
      Receive the updated subvector x

until the stopping criterion has been reached
```

The complete computing cost (T_p) for all the algorithms can by obtained by adding the arithmetic cost (T_a) plus the communication cost (T_c). When obtaining T_p we must bear in mind the time to execute one floating operation t_f. And when obtaining T_c we must take into account the time used to send a data item τ, plus the time to prepare the message (*latency*) β [4][7]. This gives us the following expression to send a message composed by *n* data items:

$$t_c = \beta + n\tau$$

A more detailed cost analysis can be found in [15]: for our algorithms the complete computing costs are:

Newton

$$T_p = k_N\left[\left(\frac{n^4}{3P}+\frac{n^3}{3P}+\frac{n^3}{3}\right)t_f+\left(n^2+n\sqrt{P}\right)\tau+(n\log_2 P+n\sqrt{P})\beta\right].$$

Chord

$$T_p = \left(\frac{n^4}{3P}+\frac{k_c n^3}{3}\right)t_f+\left(n^2+2k_c n\sqrt{P}\right)\tau+\left(n\log_2 P+k_c n\sqrt{P}\right)\beta\ .$$

Shamanksii

$$T_c = k_s\left[\left(\frac{n^4}{3P}+\frac{n^3}{3P}+\frac{mn^3}{3}\right)t_f+\left(n^2+2mn\sqrt{P}\right)\tau+\left(n\log_2 P+mn\sqrt{P}\right)\beta\right].$$

Storage cost is the same for the three methods: 1) Storing the matrices to compute *F*, and 2) Jacobian matrix and the linear system. The cost of 1 consists of storing two half sized problem matrices replicated in each processor, and the cost of 2 is storing one problem sized matrix:

$$\text{Total cost} = 2\left(\frac{n}{2}\right)^2 P+n^2 = \frac{(P+2)n^2}{2}\ ;\quad \text{Cost per processor} = \frac{n^2}{P}+\frac{n^2}{2}\ .$$

This cost improves that of the algorithms in [11] and [18] because we do not need to compute and store the eigenvectors.

4 Experimental Results

4.1 The tests

We show here a brief study of the performance of the parallel algorithm: we used a group of 15 problems [15]. Each problem consists of different kinds of *spectrum*. The three first types of spectrum are generated randomly, following some statistic distributions. The other 12 types correspond to the eigenvalues of tridiagonal matrices used as test matrices in several papers [12] [3]. In the latter 12 spectra we can distinguish between the first 7, where the elements and the spectra are generated using well defined formulas, and the last 5, where the matrices and the spectra are generated using LAPACK's `dlatms` [6] routine.
We chose here type 4 of the 15 test problems and applied the Newton's Parallel method. We used 6 different problem sizes N=200, 256, 400, 800, 1200, and 1600.

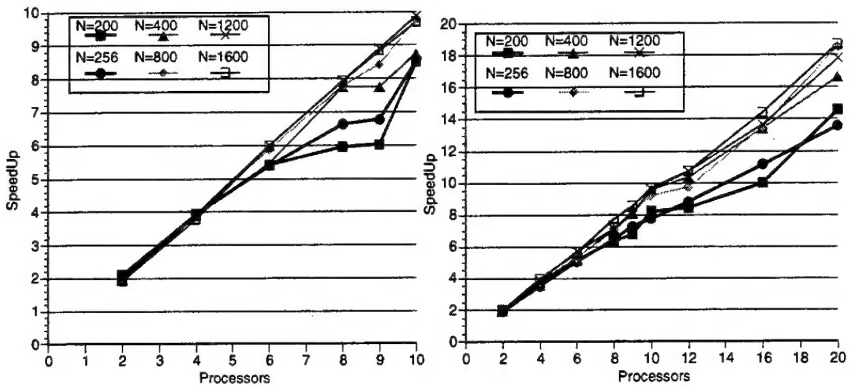


Fig. 3. Speedup figures corresponding to the SGI (left) and the cluster of PC's (right).

The experiments were carried out on two different machines: a SGI multiprocessor with 10 processors MIPS R10000/195 MHz, and on a cluster of 20 Pentium II/300 MHz PC's with a Myrinet network. The figures (Fig. 3) show the speedup of our algorithms. Speedup was obtained with respect to the Chord algorithm because it performs better than the Powell's method standard algorithm, used in the MINPACK-1 package [14].
The good performance obtained on both machines can be clearly seen. In both figures we are near the theoretical maximum speedup. The performance is good even for small problem sizes.
A scalability study was also carried out. Fig. 4 shows the scaled speedup [10]. N is the initial size for each case and is increased times a factor $k(p)$ when increasing the number of processors p (see [10]). For example for the N=200 case, the successive sizes will be 238,282,313.....

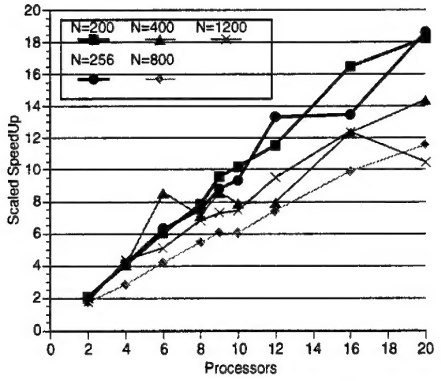


Fig. 4. Scaled speedup corresponding to the cluster of PC's.

Fig. 4 shows that our algorithms are scalable. Performance is specially good for problems where the initial sizes for the scalability test are small and medium. For large initial size scalability decreases slightly.

4.2 Performance evaluation compared with the theoretical model

In this section we carried out a theoretical performance analysis for the parallel algorithms. We used the theoretical costs shown in the former sections and a machine analysis to parameterize our machine. The analysis consists of obtaining the parameters to characterize the machine (t_f, τ, β), and with these parameters, the main goal is to obtain the theoretical behaviour of our algorithms. In our case we performed the analysis using a network of computers: standard PC's and a Myrinet network, and all our algorithms.

To obtain t_f (the flop time) we could use a standard routine of any of the sequential libraries, but this time varies too much between different routines, that is to say, for different algorithms we will obtain different flop times. Another more accurate possibility consists of obtaining t_f using our sequential algorithm. With this analysis the flop time is $t_f = 0.018$ microseconds.

To obtain the communications time, we used the double Ping-Pong algorithm where one processor sends several different sized messages to another, which then returns the messages. The measured time in this operation is half that required to send and give back each message. Sending the minimum sized packages we can obtain the value of β, while sending the maximum sized messages we can obtain the value of τ. The value obtained for the latency β is 33 microseconds, and the value for τ is 0.03 microseconds.

We included this study for two reasons: firstly, to test if the theoretical model developed before is good, and secondly, to be able to predict the behaviour of the algorithm on a computer, using the theoretical model obtained.

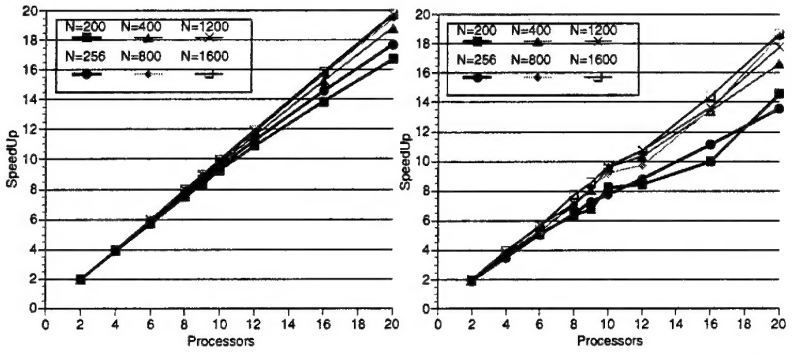


Fig. 5 and 6. Comparing the theoretical speedup model (left) and the experimental speedup (right).

We can compare the figures corresponding to the theoretical and experimental speedup models. It can be appreciated that the two figures are very similar. The only small difference corresponds to small size matrices (200 and 256). In addition the theoretical speedup is a little better than the experimental. This is normal.

With this analysis we can assume that our theoretical model is good. In principle such a model could be good to predict the behaviour of the algorithm on the computer, changing the size of the problem and/or the number of processors.

5 Conclusions

We have developed a new approach to solve the inverse eigenproblem for RST matrices. Our method has several advantages with respect to "state of the art" algorithms. We solve the problem as a general nonlinear system, using the difference approximation technique to approximate the Jacobian. *This gives a more general perspective on this problem.* We have also managed to reduce storage cost, which allows us to work with larger problems.
Furthermore, our parallel algorithm is efficient when working with small and medium sized problems.

With respect to the theoretical model, we think the model is quite close to the experimental model. This is very important because with such a model in principle we can predict at the behaviour of any algorithm using the parameterized machine (in our case the Myrinet network). Finally, we note the behaviour of the Myrinet network. We think it could be a good, cheap alternative to classical MPP machines.

6 References

1. Anderson E., Bai Z., Bischof C., Demmel J., Dongarra J., Du Croz J., Greenbaum A., Hammarling S., Mackenney A., Ostrouchov S., Sorensen D. (1995). *LAPACK Users' Guide. Second edition.* SIAM Publications, Philadelphia
2. Badía J.M., Vidal A.M. (1997). *On the bisection Method for the Computation of the Eigenvalues of Symmetric Tridiagonal Matrices.* Technical Report. DI 01-02/97. Departamento de Informática. Universidad Jaime I. Castellón.
3. Badía J.M., Vidal A.M. (1999). *Solving the Inverse Toeplitz Using SCALAPACK and MPI.* Lecture Notes in Computer Science, 1697 (1999), pp. 372-379. Ed. Springer–Verlag.
4. Blackford L.S., Choi J., Cleary A., D'Azevedo E., Demmel J. Dhillon I., Dongarra J., Hammarling S., Henry G., Petitet A., Stanley K., Walker D., Whaley R.C. (1997). *SCALAPACK Users' Guide.* SIAM Publications, Philadelphia
5. Cantoni, A. and Butler, F. (1976). *Eigenvalues and eigenvectors of Symmetric Centrosymmetric Matrices.* Lin. Alg. Appl., no. 13. pp. 275-288.
6. Demmel, J., MacKenney, A. (1992). *A Test Matrix Generation Suite, LAPACK Working Note 9.* Courant Institute, New York.
7. Dongarra J.J., Dunigan T. (1995). *Message-Passing Performance of Various Computers.* Technical Report UT-CS-95-299. Departtment of Computer Science. University of Tenesse.
8. Golub G. H. , Van Loan C. F. (1996). *Matrix Computations (third edition).* John Hopkins University Press.
9. Kelley C.T. (1995). *Iterative Methods for Linear and Nonlinear Equations.* Frontiers in Applied Mathematics, SIAM Publications, Philadelphia.
10. Kumar R ., Grama A., Gupta A., Karypis G. (1994). *Introduction to Parallel Computing: Design and Analysis of Algorithms.* The Benjamin Cumimngs Publishing Company.
11. Laurie D.P. (1988). *A numerical approach to the inverse Toeplitz eigenproblem matrix*, J. Sci. Statist.Comput., 9 (1988), pp .401-405.
12. Ly T.Y., Zeng Z. (1993*). The Laguerre iteration in solving the symmetric tridiagonal eigenproblem, revisited.* SIAM J. Sci. Statist. Compt., vol. 13, no. 5, pp 1145-1173.
13. Myrinet. *Myrinet overview.* (Online). http://www.myri.com/myrinet/overview/index.html.
14. Moré J.J., Garbow B.S., Hillstrom K.E, (1980). *User Guide for MINPACK-1.* Technical Report. ANL-80-74. Argonne National Laboratory.
15. Peinado J, Vidal A.M. (1999). *A new Parallel approach to the Toeplitz Inverse Eigenproblem using the Newton's Method and Finite Difference techniques.* Technical Report. DSIC-II/15/99. Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia.
16. Sun, X.H. (1995). *The relation of Scalability and Execution Time.* Internal Report. 95-62, ICASE NASA Langley Research Center, Hampton, VA, 1995.
17. Skjellum A. (1994). *Using MPI: Portable Programming with Message-Passing Interface.* MIT Press Group.
18. Trench, W.F. (1997). *Numerical Solution of the Inverse Eigenvalue Problem for Real Symmetric Toeplitz Matrices.* SIAM J. Sci. Comput., vol. 18, no. 6. pp. 1722-1736.
19. Whaley R.C. (1994). *Basic Linear Algebra Communication Subprograms (BLACS): Analysis and Implementation Across Multiple Parallel Architectures.* Computer Science Dept. Technical Report CS 94-234, University of Tennesee, Knoxville.